Rekursion und Iteration

1 Schleife als Rekursion ausdrücken

In `01-loop` befindet sich ein Programm mit einer Schleife, das die Zahlen von 0 bis 9 in der Variable `sum` aufsummiert. Das Hochzählen und das Addieren ist in zwei Methoden ausgelagert, `next()` und `work()`, die jeweils auch eine Ausgabe erzeugen, damit die einzelnen Iterationsschritte leicht nachzuvollziehen sind.

Die `for`-Schleife setzt `i` auf 0, führt `work()` aus und erhöht `i` mithilfe von `next()` auf 1. Dann wird geprüft ob `i < N` ist, was der Fall ist, und die nächste Iteration beginnt: zuerst `work()`, dann wird `i` mithilfe von `next()` erhöht auf 2, dann wird geprüft ob `i < N` ist, was der Fall ist, usw. Nach der zehnten Iteration hat `i` den Wert 10, der Test `i < N` ergibt `false` und die Schleife endet.

```
work(0)
next(0)
work(1)
next(1)
work(2)
next(2)
work(3)
next(3)
work(4)
next(4)
work(5)
next(5)
work(6)
next(6)
work(7)
next(7)
work(8)
next(8)
work(9)
next(9)
```

Die selbe Abfolge von `work()` und `next()` kann auch durch den Aufruf einer rekursiven Methode erreicht werden.

Die Methode `workThenNext()` soll dazu zuerst `i < N` prüfen und dann, falls diese Bedingung `true` ergibt, `work()` aufrufen. Der nächste Iterationsschritt wird erreicht indem `workThenNext()` sich selbst mit einem neuen Argument aufruft, und zwar mit dem Ergebnis von `next()`.

Der erste Aufruf von `workThenNext()` mit `i = 0` ist noch nicht zu Ende. Stattdessen wurde `workThenNext()` ein zweites Mal mit `i = 1` aufgerufen. Dieser zweite Aufruf wird seinerseits einen weiteren Aufruf von `workThenNext()` mit `i = 2` verursachen, usw, bis bei dem Aufruf in dem `i` auf 10 gesetzt wird, die Bedingung `i < N` nicht mehr erfüllt ist, die Methode ihr Ende erreicht und zu ihrem Aufruf zurückspringt.

Vervollständigen Sie `workThenNext()`, sodass für `sum` das gleiche Ergebnis berechnet wird, wie in der Schleife und die Ausgabe von `work()`/`next()` so aussieht:

```
work(0)
next(0)
  work(1)
  next(1)
    work(2)
    next(2)
      work(3)
      next(3)
        work(4)
        next(4)
          work(5)
          next(5)
            work(6)
            next(6)
              work(7)
              next(7)
                work(8)
                next(8)
                  work(9)
                  next(9)
sum = 45
```

Die Methode `nextThenWork()` soll dasselbe machen wie `workThenNext()`, jedoch in umgekehrter Reihenfolge: zuerst soll der nächste Iterationsschritt gestartet werden. Erst nachdem der rekursive Aufruf beendet ist, wird mit `work()` die eigentliche Arbeit erledigt. Diese Reihenfolge erzeugt folgende Ausgabe:

```
next(0)
  next(1)
    next(2)
      next(3)
        next(4)
          next(5)
            next(6)
              next(7)
                next(8)
                  next(9)
                  work(9)
                work(8)
              work(7)
            work(6)
          work(5)
        work(4)
      work(3)
    work(2)
  work(1)
work(0)
sum = 45
```

Bei `workThenNext()` ist die Rekursion am Ende jedes Iterationsschrittes ("tail recursion"). Bei `nextThenWork()` ist es umgekehrt ("head recursion").

Aufgaben:

- ☐ `workThenNext()` vervollständigen (andere Methoden dabei nicht verändern!)
- ☐ `nextThenWork()` vervollständigen (andere Methoden dabei nicht verändern!)
- ☐ Besprechen Sie mit der Person neben Ihnen die Unterschiede zwischen `nextThenWork()` ("head recursion") und `workThenNext()` ("tail recursion").

2 Rekursion Bilden

In `02-binary-formatter` ist ein unfertiges Programm, das Zahlen einliest und sie als Binärzahlen ausgibt. Die Methode `formatAsBase2()` deckt jedoch nur drei Bits ab. Es können somit Zahlen von 0 bis 7 dargestellt werden, größere Zahlen nicht.

Schreiben Sie die Methode `formatAsBase2()` rekursiv so um, dass beliebig große positive Integer dargestellt werden können.

Aufgaben:

- ☐ `formatAsBase2()` als *head recursive* oder *tail recursive* Methode vervollständigen (andere Methoden dabei nicht verändern!)
- ☐ mit `batchrun` testen, bis alle Testfälle erfolgreich durchlaufen

3 Substring per Rekursion

Das Programm in `03-substring` liest einen String `str` gefolgt von einem Integer `len` von der Standard Eingabe. Von dem String werden nur die ersten `len` Zeichen ausgegeben. Wenn der String weniger als `len` Zeichen enthält, wird anstelle der fehlenden Zeichen ein Stern ausgegeben. Die gewünschte Ausgabe ist in den `spec.i`-Dateien dokumentiert. Mit `batchrun` können Sie wie gewohnt die Korrektheit Ihres Programms testen.

Schreiben Sie zwei rekursive Methoden, die die ersten `len` Zeichen einer Zeichenkette liefern. Falls `len` größer ist, als `str` Zeichen hat, werden Sterne (`*`) als Füllzeichen angehängt.

Das gesamte Programm soll keine Schleifen enthalten. Eine der beiden Methoden soll *head recursive*, die andere *tail recursive* implementiert werden.

Hinweis: wenn einer der beiden Rekursionsansätze mehr Probleme macht, als der andere, kann eine Hilfsmethode mit einem zusätzlichen Parameter hilfreich sein, zB `String f(String str, int len, String accum)`.

Aufgaben:

- ☐ `substrHeadRec()` als *head recursive* Methode vervollständigen (andere Methoden dabei nicht verändern!)
- ☐ `substrTailRec()` als *tail recursive* Methode vervollständigen (andere Methoden dabei nicht verändern!)
- ☐ mit `batchrun` testen, bis alle Testfälle erfolgreich durchlaufen