- OA 1 in der Mappe ist seit Mo bearbeitbar.
- Probleme mit Mappe → Sprechstunde bei Robert → www

2.2. Handy Tarife

Problem: Berechne monatl. Telefonkosten eines Minutentarifs

- Grundgebühr: 5,95 € enthält 60 Freiminuten
- danach bis zu 120 Minuten: 11 ct/min
- danach bis zu 300 Minuten: 3 ct/min
- danach Flatrate: 17,95 €

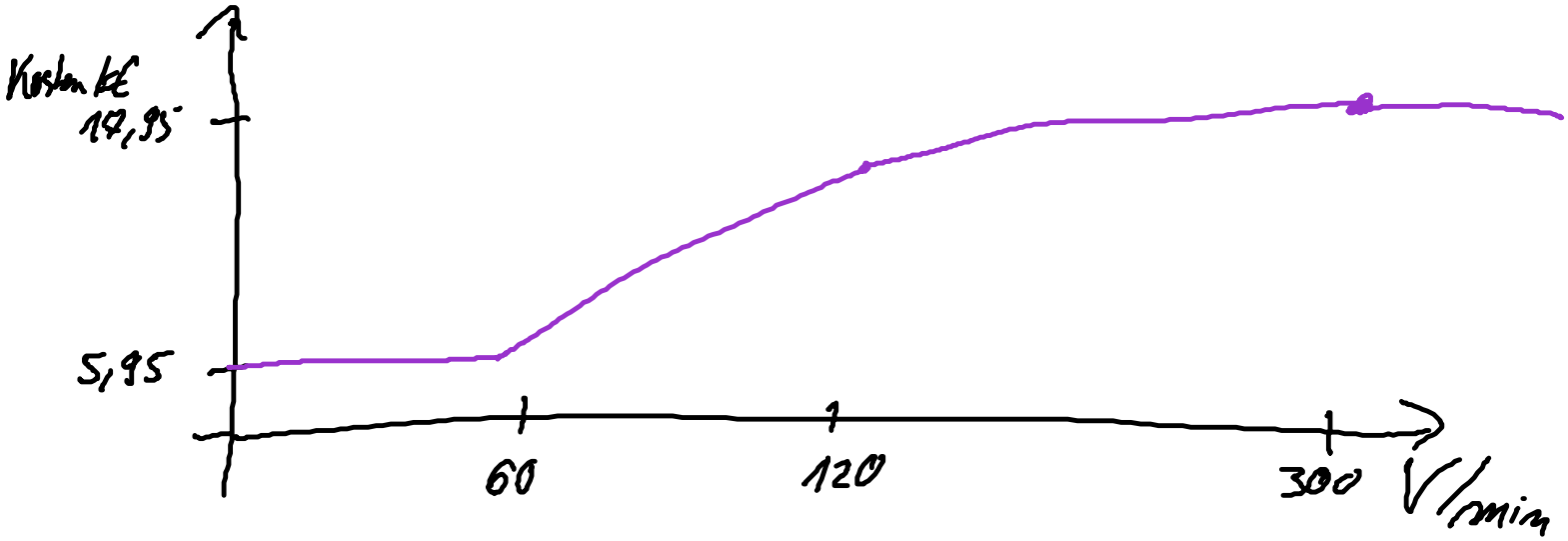
Mathematisierung: Volumen V

$V \leq 60 \rightarrow 5{,}95\,€$

$60 < V \leq 120 \rightarrow 5{,}95 + (V-60) \cdot 0{,}11\,€$

$120 < V \leq 300 \rightarrow 5{,}95 + (120-60) \cdot 0{,}11 + (V-120) \cdot 0{,}03\,€$

$300 < V \rightarrow 17{,}95\,€$



Keine geschlossene Formel, sondern Fallunterscheidung.
→ führt zu if-Anweisungen in Java

Java-Programm (ähnlich zu Temperatur.java)
auf www.

```java
// HandyTariff.java
//
// calculates the monthly fee for a handy tariff

//import scanner class for command line reading
import java.util.Scanner;

public class HandyTariff{
        public static void main(String[] args) {

        //constants for tariff calculation
        final double                    BASE_RATE = 5.95,
                     LOW_VOLUME = 60,   LOW_RATE = .11,
                     HIGH_VOLUME = 120, HIGH_RATE = .03,
                     FLAT_VOLUME = 300, FLAT_RATE = 17.95;

        int volume; // assume volume is in full minutes
        double fee;

        Scanner scan = new Scanner(System.in);
        System.out.println("Geben Sie Ihr monatliches Gesrächsvolumen "
                           + "in Minuten an und drücken Sie Return: ");
        // read command line and interpret as int
        volume =  scan.nextInt();

        // calculate fee
        if (volume <= LOW_VOLUME){  // BASE_RATE applies
            fee = BASE_RATE;
        } else if (volume <= HIGH_VOLUME) { // LOW_RATE applies
            fee = BASE_RATE + LOW_RATE * (volume - LOW_VOLUME);
        } else if (volume <= FLAT_VOLUME) { // HIGH_RATE applies
            fee = BASE_RATE + LOW_RATE * (HIGH_VOLUME - LOW_VOLUME)
                            + HIGH_RATE * (volume - HIGH_VOLUME);
        } else { // volume > FLAT_VOLUME, FLAT_RATE applies
            fee = FLAT_RATE;
        }

        // round to Eurocent
        fee = Math.round(fee * 100) / 100.0;
```

Konstanten sollte man immer über Bezeichner ansprechen. Bei möglichen Änderungen muss der Wert nur an einer Stelle geändert werden.

geschachtelte if-Anweisung

```
        //output the calculated fee
        System.out.println("Bei " + volume + " Gesprächsminuten "
                           + "zahlen Sie " + fee + " Euro.");
    }
}
```

Grundversion der if-Anweisung

```
if (Bedingung) {
    // Anweisungen, falls Bed. erfüllt
} else {
    // Anweisungen, falls Bed. nicht erfüllt
}
```

Ausdruck vom Typ boolean (d.h. Liefert Wert "false" oder "true")

else-Teil optional

if-Anweisungen können geschachtelt werden.

Dann bezieht sich **else** immer auf das letzte **if**.

Bsp:

```
double a, b;
:
if (a<0)
    b=0;
if (a>b)
    b=10;
else
    b=20;
```

Bed. false

Ergebnis:
für alle a ≤ 0
wird b auf 20 gesetzt.

Wichtig: richtig klammern!

```
if (a<0)
    b=0;
    c=1;
```

wird immer ausgeführt

möglicherweise war gemeint:

```
if (a<0) {
    b=0;
    c=1;
}
```

2.3 Primzahlen

Problem: geg. Zahl $n \in \mathbb{N}$, $n > 0$
gesucht: nächstgrößere Primzahl $p > n$

Idee: Überprüfe $n+1, n+2, \ldots$ ob sie prim sind.

Frage: Terminiert dieser Algorithmus für jedes $n \in \mathbb{N}$?

Antwort: Ja! Denn es gibt unendlich viele Primzahlen. (Satz von Euklid)

Algorithmus Primzahl (in Pseudocode)

Input: $n \in \mathbb{N}$
Output: $p \in \mathbb{N}$, kleinste Primzahl $p > n$

```
Lese n ein
z := n
REPEAT
    erhöhe z um eins // z := z+1, z++
    überprüfe, ob z eine Primzahl ist ⇝ Primzahl
UNTIL (z ist Primzahl)
gib z aus
```

Programm dazu auf Homepage!

Test, ob $z \in \mathbb{N}$ Primzahl ist:
z ist Primzahl $\Leftrightarrow$ keine Zahl $2 \leq k \leq z-1$ ist Teiler von z

Es genügt, die Zahlen $k = 2, 3, \ldots, \lceil\sqrt{z}\rceil$ zu testen, ob sie Teiler von z sind. Denn es gilt:

OE: $z \geq 5$

> z hat Teiler $(\neq 1) \Rightarrow z$ hat Teiler $k \leq \lceil\sqrt{z}\rceil$

Beweis: Sei k Teiler von z, d.h. $z = k \cdot l$, für $l \in \mathbb{N}$

Annahme: $k > \lceil\sqrt{z}\rceil$, $l > \lceil\sqrt{z}\rceil \Rightarrow z = k \cdot l > \lceil\sqrt{z}\rceil \cdot \lceil\sqrt{z}\rceil$ ↯

$\Rightarrow$ Ann. ist falsch $\Rightarrow k \leq \lceil\sqrt{z}\rceil$ oder $l \leq \lceil\sqrt{z}\rceil$. □

„Widerspruchsbeweis"

boolean divisorFound = false, int k=2;

```
while ((!divisorFound) && (k*k <= z)) {
    // check if k is a divisor of z
    if (z % k == 0) {
        divisorFound = true;
    }
    k++;
} //end while
```

} Test, ob z Primzahl

k++ ≙ k = k+1

while-Schleife in Java

```
while (Bed.) {
    // Anweisungen
}
```

werden ausgeführt, solange Bed. wahr ist (evtl. kein einziges Mal)

„abweisende Schleife"

```
do {
    z++;
    k = 2;
    divisorFound = false;  // so far no divisor of z has been found

    while ((!divisorFound) && (k*k <= z)) {
        // check if k is a divisor of z
        if (z % k == 0) {
            divisorFound = true;
        }
        k++;
    } //end while

} while (divisorFound);
```

äußere Schleife ist do-while Schleife.

```
do {
    // Anweisungen
} while (Bed.);
```

do-while $\hat{=}$ REPEAT ... UNTIL (Bed.)

Pseudocode

werden solange ausgeführt, wie Bed. wahr ist, auf jeden Fall einmal.

„ akzeptierende Schleife "