13 Implementierung einfacher Methoden

```java
class BasicMethods {
    public static void main(String[] args) { ... }

    static int abs(int x) {
        return x < 0 ? x * -1 : x;
    }

    static int min(int[] values) {
        int smallest = Integer.MAX_VALUE;

        for (int val : values)
            if (val < smallest)
                smallest = val;

        return smallest;
    }

    static double average(double[] values) {
        double sum = 0;

        for (double val : values)
            sum += val;

        return values.length == 0 ? 0 : sum / values.length;
    }

    static boolean exists(int[] values, int x) {
        for (int val : values)
            if (val == x)
                return true;

        return false;
    }
}
```

```
$ java BasicMethods
|-5| = 5
min[1, 2, 3, 4, -5] = -5
avg[1.1, 2.2, 3.3, 4.4] = 2,750000
-5 is in [1, 2, 3, 4, -5]
```

14 CaesarCipher mit Methoden

```java
public class CaesarCipher {
    static final int ALPHABET_SIZE = 26;

    public static void main(String[] args) {
        char ch;
        do {
            Out.print("'e' for encrypt, 'd' for decrypt, 'q' to quit: ");
            ch = In.readChar();

            if (ch == 'e' || ch == 'd') {
                Out.print("Filename: ");
                String filename = In.readWord();
                In.readLine();

                Out.print("Key: ");
                int key = In.readInt() % ALPHABET_SIZE;
                In.readLine();

                Out.print("Text: ");
                String text = getText(ch == 'd' ? filename : null);

                if (ch == 'e') {
                    Out.open(filename);
                }

                String cipher = transformText(text, ch == 'e' ? key : -key);

                Out.print(cipher);

                if (ch == 'e') {
                    Out.close();
                } else {
                    Out.println();
                }
            }
        } while (In.done() && ch != 'q');
    }

    public static String transformText(String text, int key) {
        StringBuilder cipher = new StringBuilder(text);

        for (int i = 0; i < cipher.length(); i++) {
            char curChar = cipher.charAt(i);

            if (Character.isLetter(curChar)) {
                cipher.setCharAt(i, transformChar(curChar, key));
            }
        }

        return cipher.toString();
```

```
51      }
52
53      public static char transformChar(char ch, int key) {
54          int lbound, ubound;
55
56          if (Character.isLowerCase(ch)) {
57              lbound = 'a';
58              ubound = 'z';
59          } else {
60              lbound = 'A';
61              ubound = 'Z';
62          }
63
64          char newChar = (char) (ch + key);
65          if (newChar > ubound)
66              newChar -= ALPHABET_SIZE;
67          if (newChar < lbound)
68              newChar += ALPHABET_SIZE;
69
70          return newChar;
71      }
72
73      public static String getText(String filename) {
74          String text;
75
76          if (filename != null) {
77              In.open(filename);
78              text = In.readFile();
79              In.close();
80          } else text = In.readLine();
81
82          return text;
83      }
84  }
```

```
$ java CaesarCipher
'e' for encrypt, 'd' for decrypt, 'q' to quit: e
Filename: test.txt
Key: 5
Text: Hello world.
'e' for encrypt, 'd' for decrypt, 'q' to quit: q
$ cat test.txt
Mjqqt btwqi.
$ java CaesarCipher
'e' for encrypt, 'd' for decrypt, 'q' to quit: d
Filename: test.txt
Key: 31
Text: Hello world.
'e' for encrypt, 'd' for decrypt, 'q' to quit: q
```

15 Rekursive Taylorreihe

```java
class RecursiveTaylor {
    public static void main(String[] args) { ... }

    static double expTaylor(double x, int i) {
        if (i == 0)
            return 1;

        return (1 / factorial(i)) * Math.pow(x, i) + expTaylor(x, i - 1);
    }

    static double factorial(double n) {
        if (n <= 1)
            return 1;

        return n * factorial(n - 1);
    }
}
```

```
$ java RecursiveTaylor
x: 5
i_max: 12

x: 5,000000
expTaylor: 148,113535
Math.exp: 148,413159
diff: 0,299624
```

16 Rekursive `drawLine`-Implementierung

```
class RecursiveLine {
    public static void main(String[] args) { ... }

    static void drawLine(int x1, int y1, int x2, int y2, int radius,
        int distance)
    {
        double dist = Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));

        if (dist < distance) {
            Window.drawCircle(x1, y1, radius);
            Window.drawCircle(x2, y2, radius);

            return;
        }

        double midx = x1 + ((x2 - x1) / 2);
        double midy = y1 + ((y2 - y1) / 2);

        drawLine(x1, y1, (int)midx, (int)midy, radius, distance);
        drawLine((int)midx, (int)midy, x2, y2, radius, distance);
    }
}
```