Rekursion Übungen 1

(A1) Potenzberechnung

Implementiere eine rekursive Methode `potenz(a,n)`, die bei Eingabe einer Dezimalzahl a und einer natürlichen Zahl n als Ergebnis die Potenz a^n zurückgibt.

Beispiel: Der Aufruf `potenz(2.5,3)` gibt den Wert 15,625 zurück.

Hinweis 1

Beginne mit einem Methodengerüst, welches eine Verzweigung enthält, die den Basisfall und den Rekursionsfall unterscheidet.

Was ist der Basisfall - also wann weißt du sofort, was die Potenz ist, ohne zu überlegen (unabhängig von der Basis, betrachte den Exponenten).

Hinweis 2: Pseudocode

```
potenz(double basis, int exponent):
  wenn exponent ist gleich 0:
     return 1
  sonst
     return basis * potenz(basis, exponent-1)
```

Hinweis 3: Methodengerüst mit Basisfall

```
public double potenz(double b, int e)
    {
        if(e==0) {
            return 1;
        } else {
            // Rekursionsfall ??
        }
    }
```

Was muss im Rekursionsfall berechnet werden? Welchen Aufrufparameter muss man beim rekursiven Aufruf der Methode verändern, damit der Basisfall irgendwann erreicht wird?

(A2) Verzinsung

Implementiere eine rekursive Methode `guthaben(g,z,a)`, die bei Eingabe eines Guthabens `g` in Euro, eines Zinssatzes `z` in Prozent und einer Laufzeit `a` in Jahren als Ergebnis das verzinste Guthaben nach Ende der Laufzeit zurückgibt.

Beispiel: Der Aufruf `guthaben(1000,1,2)` gibt den Betrag 1020,10 (€) zurück.

Hinweis

Das funktioniert genau wie Aufgabe 1, wenn man sich klar macht, dass das Guthaben nach a Jahren berechnet werden kann als:

```
G(g,z,a)=g*(1+(z/100))^a
```

(A3) Fibonacci-Zahlen

Implementiere eine rekursive Methode `fibonacci(n)`, die bei Eingabe einer natürlichen Zahl `n` als Ergebnis die n-te Fibonacci-Zahl zurückgibt. Die erste und zweite Fibonacci-Zahl ist jeweils 1. Die weiteren Fibonacci-Zahlen berechnen sich als Summe der beiden Vorgängerzahlen. Die ersten zehn Fibonacci-Zahlen lauten: `1, 1, 2, 3, 5, 8, 13, 21, 34, 55`.

Beispiel: Der Aufruf `fibonacci(11)` gibt die Zahl 89 zurück.

Hinweis

Das ist ein Beispiel, in dem sich die Funktion im Rekursionsfall mehr als einmal selbst aufruft. Es muss ja `fibonacci(n-1)` und `fibonacci(n-2)` bekannt sein, um `fibonacci(n)` auszurechnen.

Fange an wie immer: Fallunterscheidung, Basisfall. Überlege dann, was im Rekursionsfall geschehen muss.

Methodengerüst

```
public int fibonacci(int n){
    // Bsisfall
    if (n==1||n==2) {
      return FIXME
    } else {
      return FIXME
    }
```

```
}
```

(A4) Palindrom

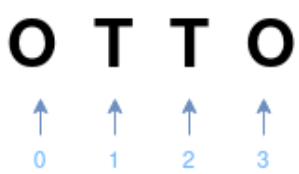
Palindrome sind Wörter wie OTTO oder RELIEFPFEILER, die vorwärts wie rückwärts gelesen gleich sind. Implementiere eine rekursive Methode `palindrom(text,l,r)`, die bei Eingabe eines Strings `text` sowie einer linken Feldgrenze `l` und einer rechten Feldgrenze `r` überprüft, ob `text[l..r]` ein Palindrom ist. Das Ergebnis der Methode soll ein Wahrheitswert sein.

Beispiele:

- Der Aufruf `palindrom("OTTO", 0, 3)` gibt `true` zurück.
- `palindrom("ORTO",0,3)` gibt `false` zurück.

Hinweise & Tipps

Einen String kann man sich in Java als Array von Char-Werten vorstellen. Auf einzelne Buchstaben kann man mit der Methode `charAt()` des String-Objekts zugreifen. Wie bei allen Arrays beginnt die Zählung bei 0.




```
word="OTTO";
char zeichen = word.charAt(2); // zeichen ist 'T'
int laenge = word.length(); // laenge ist 4
```

Die Länge des Strings kann man mit `word.length()` ermitteln.

Tipp 1: Basisfall

Überlege dir zunächst, wann der Basisfall eintritt: Für welche Worte kannst du sofort ohne nachdenken sagen, dass Sie ein Palindrom sind? Wie hängt diese Eigenschaft mit den Parametern `start` und ende zusammen?

Tipp 2: Rekursionsfall

Anders als bei den bisherigen Beispielen muss sich die Methode nicht unbedingt wieder selbst aufrufen, sondern nur dann, wenn das Wort nach dem bisherigen Kenntnisstand ein Palindrom sein könnte. Welche Bedingung muss erfüllt sein, damit es sich lohnt, das Wort weiter zu untersuchen?

Hinweis: Codegerüst

```
public boolean palindrom(String word, int start, int end)
    {
        // Basisfall
        if (FIXME) {
            return FIXME;
        }
        // Rekursionsfall
        if (FIXME) {
            // Was muss hier alles geschehen?
        }
        // Kein Palindrom
        return false;
    }
```

Lösungsvorschlag 1

```
public boolean palindrom(String word, int start, int end)
    {
        // Basisfall
        if (end-start<=0 ) {
            return true;
        }
        // Rekursionsfall
        if (word.charAt(start) == word.charAt(end)) {
            start=start+1;
            end=end-1;
            return palindrom(word, start, end);
        }
        // Kein Palindrom
        return false;
    }
```

Lösungsvorschlag 2

```
public boolean palindrom(String word, int start, int end) {

    // Basisfall
    if (end - start <= 1) {
        return word.charAt(start) == word.charAt(end);
    }
    // Rekursionsfall
    else {
        start = start + 1;
        end = end - 1;
```

```
        return palindrom(word, start, end);
    }
```