Aufgabe 1

Betrachte zur Wiederholung die folgende **rekursive Methode**:

```
01  int fibonacci(int n)
02  {
03     if (n == 1 || n == 2) return 1;
04     else                  return fibonacci(n-1) + fibonacci(n-2);
05  }
```

a) Erstelle ein Ablaufprotokoll des Methodenaufrufs **`a = fibonacci(5)`**.
Gib dabei nur die Zeilennummern an, die tatsächlich ausgeführt werden, jeweils mit den Werten für den Parameter n und den Rückgabewert.

Als Hilfestellung die ersten Schritte:

```
01 fibonacci(5)
   01 fibonacci( ? )
   ...

   01 fibonacci( ? )
   ...

   04 return ?
```

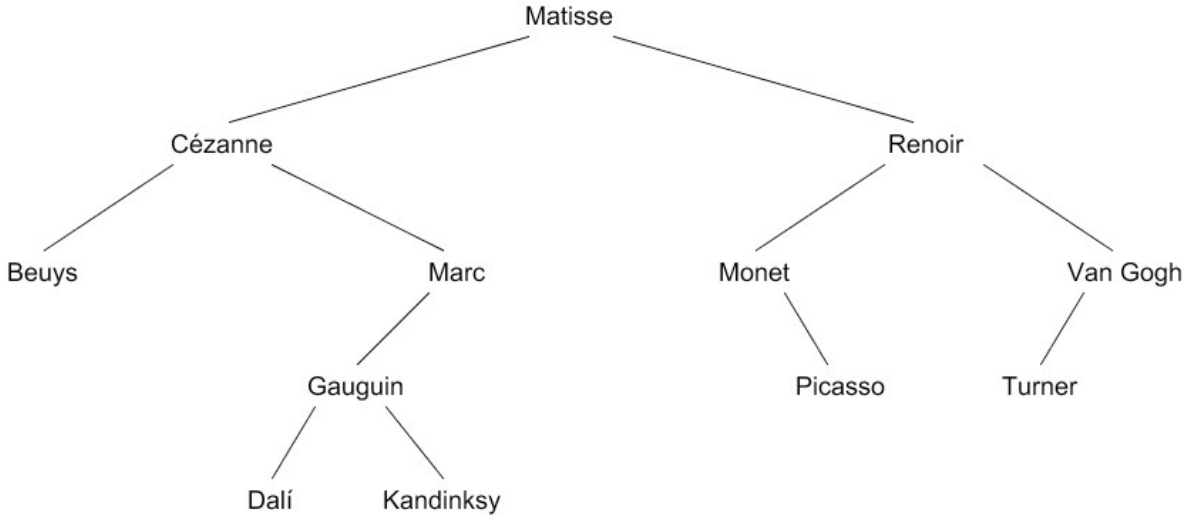
Der Aufruf beginnt mit fibonnaci(5).
Zeile 03 wird nicht ausgeführt, da n ≠ 1 und 2.

In Zeile 4 wird fibonnaci() wieder aufgerufen, daher springt der Ablauf zweimal zu Zeile 1.

Erst danach wird der Rückgabewert berechnet.

b) Welchen Wert erhält **`a`** am Ende?

Aufgabe 2



Die **Suche im binären Suchbaum** kann auch rekursiv programmiert werden.
Für den abgebildeten Suchbaum wird die Methode suche() (siehe nächste Seite) aufgerufen.

```
01  boolean suche(String suchName, BinaryTree<Adresse> knoten)
02  {
03     if (knoten.isEmpty())
04     {
05        return false;
06     }
07     String knotenName = knoten.getContent().getName();
08     if (suchName.equals(knotenName))
09     {
10        return true;
11     }
12     if (suchName.compareTo(knotenName) < 0)
13     {
14        return suche(suchName, knoten.getLeftTree());
15     }
16     else
17     {
18        return suche(suchName, knoten.getRightTree());
19     }
20  }
```

a) Beschreibe den Ablauf der folgenden Methodenaufrufe schrittweise.
Gib nur die fettgedruckten Zeilennummern an, in der Reihenfolge, in der sie ausgeführt werden.
Rücke ein, um anzudeuten, welche Zeilen zu welchem Aufruf gehören.
Für jeden Aufruf von suche() gib an, auf welchen Namen der Parameter „knoten“ zeigt.

```
boolean b1, b2;

b1 = suche("Kandinsky", wurzel);

b2 = suche("Warhol", wurzel);
```

b) Welche Werte haben b1 bzw. b2 anschließend?