# Tutorial #5. Recursion and search

## Stop rule

There are **two types** of recursion: finite and infinite. For finite one you have "**stop rule**" – condition in which we know the result of the function ("**base case**" in lecture). Infinite recursion is useless in programming but useful in math – (representing different recurring sums and continued fractions). For calculating infinite math formulas, we can use stop rule for the sake of accuracy.

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \ddots}}}$$

```
public static double squareRoot2(int deep) {
      if (deep == 22) return 1;
      // deep == 10 - diff in 10th sign after point
      // deep == 20 - diff in 16th sign after point
      // deep == 22 - equal in double
      double value = 1 / (2 + squareRoot2(deep + 1));
      if (deep == 0) value += 1;
      return value;
}
```

## Recursion and stack

Inside recursion there always lives a Stack. This is a part of programming paradigm where we can use subprograms. When we call one method from another, Java **push()** our **return address** and local data (maybe call parameters in some implementations) into stack. When this invoked methods ends, it **pop()** this data to know, what to do next (where to return). In Java we cannot access this stack directly, but we can view it using `Thread.dumpStack();`

## Practice

1) Implement **recursive binary search** (reference lecture slides). Use **Thread.dumpStack**() to see the stack state on each iteration. It should return the **index of the element** or -1 otherwise.
2) Convert this implementation to **loop+stack** based and then to **loop** based. See what has changed in your code and why.
3) You are given a template of class structure (*Table, Row, Index*, etc.). Study this template. Run tests in **Tests** class. Learn how to interact with the database using classed and SQL. Study SQL limitations.
4) Uncomment Line 22 in **Tests** class. What happened?
5) Run tests in **MyFramework** class. See the output.
6) Embed your search implementation into **MyFramework** class methods **binarySearch**(…) and **binarySearchOrNext**(…). ***binarySearchOrNext***() *– it should return element index if present, else – index of next greater element.*
7) Run tests in **MyFramework** class again. See the output. What happened?