CODEGYM

Java developer in 12 months

Mentor-supported training program from CodeGym

MODULE 2. JAVA CORE

Lesson #10 Recursion

Lesson plan

- Recursion
- StackOverflowError

CODEGYM

Recursion

With the help of recursion, you can pack large and complex structures into small and simple ones, and then unfold them back when necessary.

Recursive methods in Java are methods that call themselves.

```
public static void countDown(int x){
    if (x <=0)
        System.out.println("Boom!");
    else {
        System.out.println(x);
        countDown(x - 1);
    }
}
```

What happens to variables?
With their values?
And how do we exit the method? Or do we exit all methods at once?

It's all very simple.
Imagine that a method that calls itself has been propagated many times:

Example on the next slide

Recursive call of a method

```java
public static void main(String[] args) {
    countDown(3);
}

public static void countDown(int x) {
    if (x <= 0) System.out.println("Boom!");
    else {
        System.out.println(x);
        countDown(x - 1);
    }
}
```

What “really” happens

```java
public static void main(String[] args) {
    countDown1(3);
}
public static void countDown1(int x) {
    if (x <= 0) System.out.println("Boom!");
    else {
        System.out.println(x);
        countDown2(x - 1);
    }
}
public static void countDown2(int x) {
    if (x <= 0) System.out.println("Boom!");
    else {
        System.out.println(x);
        countDown3(x - 1);
    }
}
public static void countDown3(int x) {
    if (x <= 0) System.out.println("Boom!");
    else {
        System.out.println(x);
        countDown4(x - 1);
    }
}
public static void countDown4(int x) {
    if (x <= 0) System.out.println("Boom!");
    else {
        System.out.println(x);
        countDown5(x - 1);
    }
}
```

Each time a method is called (even itself), new variables are created to hold the data for that method. There are no shared variables.

With each call, another copy of the method arguments appears in memory, but with new values. When returning to an old method its variables are used.
In other words, when recursing, in fact we call another method, but with the same code as ours!

Why do we need recursion?

There are a lot of tasks that are broken down into separate subtasks that are identical to the original task.

For example, you need to traverse all the elements of an XML tree. Each element can have several child elements, and they have their own.

Or you may need to display a list of files that are in the directory and all its subdirectories. Then you write a method that displays the files of the current directory, and then to get the files of all subdirectories you call this method, but with a different parameter - subdirectories.

CODEGYM

Display all files of a directory and its subdirectories

```
public static void main(String[] args){
    printAllFiles(new File("c:/windows/"));
}

public static void printAllFiles(File dir){
    for (File file : dir.listFiles()){
        if (file.isDirectory())
            printAllFiles(file);
        else

System.out.println(file.getAbsolutePath());
    }
}
```

Conditions of exiting recursion

A well-written recursive function must ensure that after a finite number of recursive calls, the recursion termination condition is met, causing the chain of successive recursive calls to break and return.

CODEGYM

StackOverflowError

StackOverflowError simply signals that there is no more memory.

If we have such a method:

```
int myMethod(){
  // your code
  myMethod();
}
```

In the given code, myMethod() will keep calling itself, deeper and deeper, and when the space (call stack) is full, we will get a StackOverflowError.

A common cause of a stack overflow is a bad recursive call. Typically, this is because recursive methods don't have the right termination condition, so the method ends up calling itself endlessly.

CODEGYM

Homework

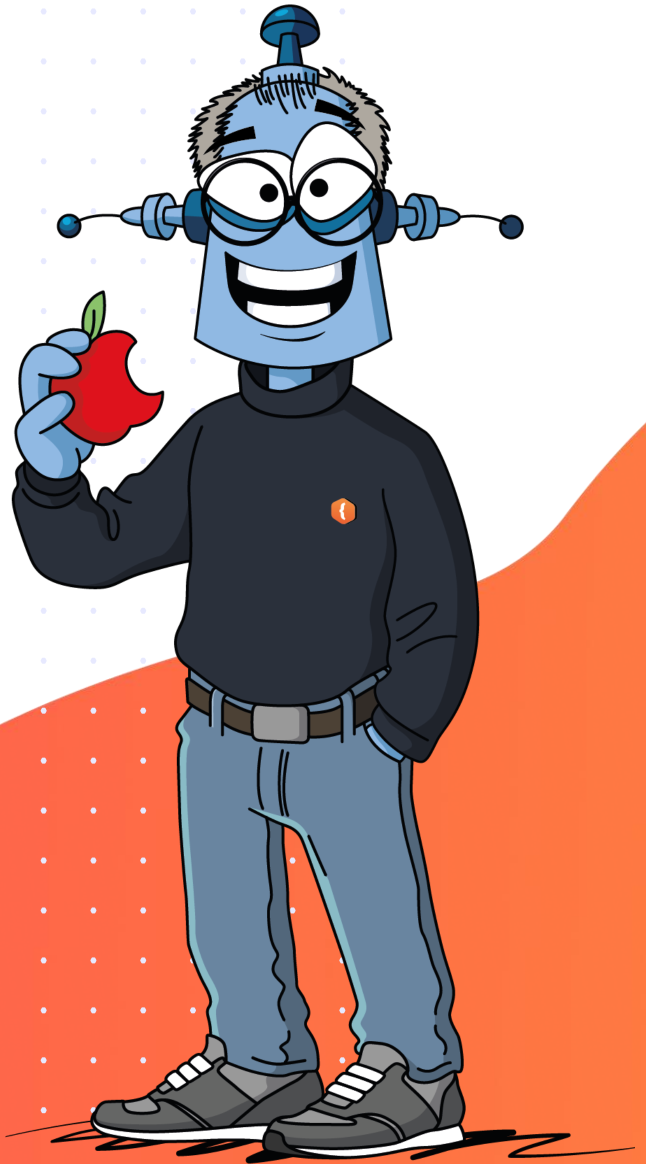
Module 2

Level 10 Recursion

MODULE 2. JAVA CORE

CODEGYM

Questions and Answers



MODULE 2. JAVA CORE