Возвратимся из реального мира к будням программиста. Простое определение: **рекурсивные функции в java** – это функции, которые вызывают сами себя. Приведу очень простой и очень вредный пример:

**public** **void** recursionFucn() {

System.out.println("Привет, JavaRush!");

recursionFucn();

}

Чем же он вреден? Предлагаю проверить самостоятельно! Если решишься на эту авантюру (а ты, скорее всего, решишься, тыжпрограммист!), напиши в комментарии, какую ошибку выдаст JVM =) И этот пример, и множество других, демонстрируют, что к применению рекурсии в Java нужно подходить осторожно. Нужно понимать, где, когда и почему оправдан такой подход к решению задачи, и следить за тем, чтобы твоя функция не превратила выполнение программы в «День Сурка». **Есть еще два важных для понимания рекурсии определения:**

* **Базис рекурсии** – условие выхода из блока рекурсивных вызовов – базисное решение задачи, при условиях, когда нет необходимости вызывать рекурсию.
* **Шаг рекурсии** – вызов функцией самой себя при изменении параметров.

Классический пример применения рекурсивной функции – вычисление факториала от числа. Если вдруг забыл, напомню: факториал положительного целого числа N (обозначается как N!) — это произведение чисел от 1 до N: N! = 1 \* 2 \* 3 \* … (N - 1) \* N. Кстати, факториал нуля по определению равен 1. Так что факториал можно найти для любого целого положительного числа и нуля (на языке математики — для множества неотрицательных целых чисел или же для множества натуральных чисел и нуля). Думаю, ты понимаешь, что запрограммировать факториал можно с помощью циклов. Собственно, вот нерекурсивный метод решения этой задачи:

**private** **int** fact(**int** n) {

**int** result = 1;

**for** (**int** i = 1; i <= n; i++) {

result = result \* i;

}

**return** result;

}

Добавим проверку того, что число положительное и целое, и отдельно проверку для нуля.

**private** **int** fact(**int** n) {

**if** (n < 0) {

System.out.println("Зачем тебе факториал из отрицательного числа?");

**return** **null**;

}

**int** result = 1;

**if** (n == 0) {

**return** result;

}

**for** (**int** i = 1; i <= n; i++) {

result = result \* i;

}

**return** result;

}

Теперь приведу код метода для рекурсивного решения этой задачи:

**private** **int** factorial(**int** n) {

**int** result = 1;

**if** (n == 1 || n == 0) {

**return** result;

}

result = n \* factorial(n-1);

**return** result;

}

Посмотрим результаты вывода для вызовов:

System.out.println(factorial(0));

System.out.println(factorial(1));

System.out.println(factorial(2));

System.out.println(factorial(3));

System.out.println(factorial(4));

System.out.println(factorial(5));

System.out.println(factorial(6));

Получим ожидаемые значения:

1

1

2

6

24

120

720

Добавим красивый вывод и вычислим фаториал для числа побольше:

**private** **int** factorial(**int** n) {

**int** result = 1;

**if** (n == 0) {

System.out.print(" = ");

**return** result;

}

**if** (n == 1) {

System.out.print(" \* 1 = ");

**return** result;

}

System.out.print(n);

**if** (n != 2) {

System.out.print(" \* ");

}

result = n \* factorial(n-1);

**return** result;

}

System.out.println(factorial(15) + "!");

Получим: 15 \* 14 \* 13 \* 12 \* 11 \* 10 \* 9 \* 8 \* 7 \* 6 \* 5 \* 4 \* 3 \* 2 \* 1 = 1 307 674 368 000! В данном случае применение рекурсивной функции оправдано и безопасно. Мы четко обозначили условие выхода из рекурсивного блока, и мы уверены в том, что он достижим: мы вводим целое неотрицательное число, в случае, если число равно нулю или единице - возвращаем результат, если же число больше — умножаем результат на функцию от числа n-1. На примере факториала от трех:

factorial(3) внутри себя выполнит следующее:

result = 3 \* factorial(2); (рекурсивный вызов)

factorial(2) внутри себя выполнит следующее:

result = 2 \* factorial(1); (рекурсивный вызов)

factorial(1) вернет 1 (базис рекурсии)

factorial(2) вернет 2 \* 1

factorial(3) вернет 3 \* 2 \* 1

По поводу осторожности применения: в чем уязвимость этой функции? Если дать методу в качестве параметра отрицательное число, то проверка

**if** (n == 1 || n == 0) {

**return** result;

}

не имеет смысла и мы уйдем в бесконечный цикл вызовов методом самого себя. Стоит добавить проверку на неотрицательность:

**if** (n < 0) {

System.out.println(«Зачем тебе факториал отрицательного числа?»);

**return** **null**;

}

И все будет хорошо.

чем преимущество одного метода перед другим? Кажется, что большой разницы нет, но на самом деле множество рекурсивных вызовов негативно скажется на производительности и потребляемой памяти: стек вызовов – практически неконтролируемый ресурс и при разных условиях вызова одной и той же рекурсивной функции, мы можем получить или не получить проблемы, связанные с этим ресурсом. Практически все задачи, решаемые с помощью итераций (циклов типа for-each), можно решить и рекурсивно. Преимущество рекурсии в читаемости и простоте написания, о недостатках мы говорили выше: возможность «выстрелить себе в ногу» неиллюзорна. Еще более осторожным надо быть при использовании так называемой «сложной рекурсии»: Функция A() вызовет функцию B(), вызывающую функцию A().Для решения таких задач необходимо полное понимание того, как работает рекурсия. Пример такой задачи: вычисление значения x^n/(n!). Факториал, как мы обсуждали выше, определен на множестве неотрицательных целых чисел. Напоследок приведу код решения. Сложная рекурсия будет состоять из двух методов:

**private** **double** calculate(**int** x, **int** n) {

**return** power(x, n) / n;

}

**private** **double** power(**int** x, **int** n) {

**if** (n == 1) **return** x;

**return** x \* calculate(x, n - 1);

}

Для входа в рекурсию используется метод calculate, вызывающий метод power, в свою очередь вызывающий метод calculate. Базис рекурсии мы обозначили в методе power:

**if** (n == 1) **return** x;

Там же определен и шаг рекурсии:

**return** x \* calculate(x, n - 1);

Осталось добавить проверку валидности входных данных:

* Любое число, кроме нуля, в нулевой степени равно 1. Если n = 0, то n! = 1. Нужно вернуть 1.
* Нуль в любой степени равен нулю.
* Неопределенность типа 0^0 рассматривать не будем и примем такие входные данные за невалидные.

Со всеми проверками методы будут выглядеть так:

**private** **double** calculate(**int** x, **int** n) **throws** ArithmeticException {

**if** (x == 0 && n == 0) {

**throw** **new** ArithmeticException("Невалидные входные данные: Неопределенность типа 0^0");

}

**if** (n < 0) {

**throw** **new** ArithmeticException("Невалидные входные данные: Факториал из отрицательного числа!");

}

**if** (n == 0) {

**return** 1;

}

**if** (x == 0) {

**return** 0;

}

**if** (x == 0) {

**return** 0;

}

**return** power(x, n) / n;

}

**private** **double** power(**int** x, **int** n) {

**if** (n == 1) **return** x;

**return** x \* calculate(x, n - 1);

}

Ну, и при вызове функции нужно не забыть поймать ошибку:

**try** {

System.out.println(calculate(x, n));

} **catch** (ArithmeticException e) {

System.out.println(e.getMessage());

}