05.
МЕТОДЫ
(ФУНКЦИИ)
((ПРОЦЕДУРЫ))

Лекции по информатике для студентов
первого курса Высшей школы ИТИС
2019 год

МИХАИЛ АБРАМСКИЙ
старший преподаватель
Высшая школа ИТИС КФУ

Мотивация

```
for (int i = 0; i < n; i++) {
     System.out.println('0');
}
for (int i = 0; i < 100; i++) {
     System.out.println('a');
}
for (int i = 0; i < 200; i++) {
     System.out.println('1');
}
for (int i = 0; i < n / 2; i++) {
     System.out.println('%');
}
```

Цели использования

- Устранение дублирование кода
- Структуризация, реализация TOP-DOWN подхода
  - Помните, писали //TODO?
  - А теперь можно сразу метод/функцию писать.
- Удобство чтения (инкапсуляция)
  - Если написан вызов факториала, то зачем мне лезть в его реализацию?

Вынужденные ограничения **этой лекции**

- Все функции объявляем рядом с main, в том же классе
- У всех них ставим public static
- В рамках данной лекции «метод» и «функция» – *синонимы*;

Разница между функцией и методом

- **Метод – функция у класса.**
- Ждем, ждем, ждем ООП

Объявление

```
public static int factorial(int n) {
    int p = 1;
    for (int i = 1; i <= n; i++) {
        p *= i;
    }
    return p;
}
```

```
public static boolean arrayHasZero(int [] array) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == 0)
            return true;
    }
    return false;
}
```

Процедуры и функции

- Разница?

- В Java – все функции.
  - void поможет

Локальные переменные

```
public static int factorial(int n) {
    int p = 1;
    for (int i = 1; i <= n; i++) {
        p *= i;
    }
    return p;
}
```

Ничего общего у обоих i нет

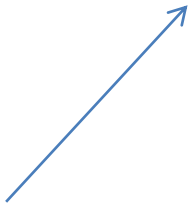
```
public static boolean arrayHasZero(int[] array) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == 0)
            return true;
    }
    return false;
}
```

Локальные переменные существуют лишь в методе

main – тоже метод,
его переменные - локальные

Вызов метода. Смысл return

```
public static void main(String[] args) {
    int [] a = new int[20];
    // Ввод массива был
    System.out.println(arrayHasZero(a));
}
```



main приостановился
управление передалось в метод arrayHasZero
когда он выполнится, оно ***вернется*** в это место
с чем вернется?
main продолжит работу

Вас вызывают!

```
public class MyClass {

    public static void f(){
        System.out.println("f");
    }
    public static void g(){
        f();
        System.out.println("g");
    }
    public static void h(){
        g();
        System.out.println("h");
    }
    public static void main(…) {
        h();
        System.out.println("main");
    }
}
```

Трасса:

- main вызвал h и ждет его конца
- h вызвал g и ждет его конца
- g вызвал f и ждет его конца
- f выполнился
- теперь g продолжил работать
- теперь h продолжил работать
- теперь main продолжил работать

main

- запустился первым,
- закончил работать последним

f

- запустился последним
- закончил работать первым

Стек вызовов

Множество вызовов – ***стек***:

- Это коллекция объектов, в которую можно добавлять и из которой удалять элементы можно только с одного конца.
- Метафоры:
  - » Обойма
  - » Парковка в узком длинном тупике

Стеки сами используются для построения алгоритмов, но это – следующий семестр.

Ошибемся – поздороваемся со стеком вызовов

```
public static void f() {
    System.out.println("f");
    int x = 10 / 0;
}
public static void g() {
    f();
    System.out.println("g");
}
public static void h() {
    g();
    System.out.println("h");
}
public static void main(String[] args) {
    h();
    System.out.println("main");
}
```

Красотища

```
Exception in thread "main" f
java.lang.ArithmeticException: / by zero
        at MyClass2.f(MyClass2.java:32)
        at MyClass2.g(MyClass2.java:36)
        at MyClass2.h(MyClass2.java:41)
        at MyClass2.main(MyClass2.java:46)
        at
sun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodA
ccessorImpl.java:62)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegatin
gMethodAccessorImpl.java:43)
        at
java.lang.reflect.Method.invoke(Method.java:497)
        at
com.intellij.rt.execution.application.AppMain.main(AppMai
n.java:144)
```

Красотища

```
Exception in thread "main" f
java.lang.ArithmeticException: / by zero
        at MyClass2.f(MyClass2.java:32)
        at MyClass2.g(MyClass2.java:36)
        at MyClass2.h(MyClass2.java:41)
        at MyClass2.main(MyClass2.java:46)
        at
sun.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodA
ccessorImpl.java:62)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(Delegatin
gMethodAccessorImpl.java:43)
        at
java.lang.reflect.Method.invoke(Method.java:497)
        at
com.intellij.rt.execution.application.AppMain.main(AppMai
n.java:144)
```

Какая-то ошибка...(((

Параметры метода

- Нет «var», «&», «*» и др.
- Все примитивные типы передаются как *параметры-значения* (их значение копируется в вызов метода).
- Все ссылочные типы данных – *по ссылке, параметр-переменная*
  - При этом очевидно, сама ссылка («адрес») ведет себя как параметр значение

Пример

```
public static void inc(int x) {
    x += 1;
}

public static void main(String[] args) {
    int x = 2;
    inc(x);
    // x = ?
}
```

Пример

```
public static void inc(int x) {
    x += 1;
}

public static void main(String[] args) {
    int x = 2;
    inc(x);
    // x = ?
}
```

верно, не сработает

Процедура

```
public static void fillArrayByRandomIntegers(int[] array) {
    Random random = new Random();
    for (int i = 0; i < array.length; i++) {
        array[i] = random.nextInt();
    }
}
```

Обратите внимание

Для ссылочных типов void методы могут быть функциональны для изменения содержимого

Но

```
public static void newArray(int [] a) {
    int [] b = new int[]{1,2,3};
    a = b;
}

public static void main(String[] args) {
    int [] z = new int[]{1,1,1};
    newArray(z);
    // z[2] = ?
}
```

Примитивы по ссылке. В Java такого нет, поэтому C#

- По значению

```
static int Max(int a, int b)
{
    return a > b ? a : b;
}
…
int y = Max(n, 10);
```

- По ссылке

```
static void Swap(ref int a, ref int b)
{
    int t = a;  a = b;  b = t;
}
…
Swap(ref x, ref y);
```

Слайд из лекции А.А.Марченко

Сигнатура метода

Название

список типов параметров

- ~~возвращаемый тип~~ (не входит!)

По сигнатуре в момент вызова определяется, какой метод использовать (есть ли он вообще)

Перегрузка метода

Методы должны различаться по сигнатуре! По названию – не обязательно!

Перегрузка - объявление методов с одинаковыми именами, но разными наборами параметров

- Работает потому, что сигнатуры разные

Пример

```
public static double difference(double a, double b) {
    return Math.abs(a - b);
}

public static double difference(double a, double b, double e) {
    double result = Math.abs(a - b);
    return result > e ? result : 0;
}

public static void main(String[] args) {
    System.out.println(difference(2.01, 2.0));
    System.out.println(difference(2.01, 2.0, 0.00001));
}
```

Такая перегрузка не айс

```
public static void f1(double a) {
    System.out.println("double");
}

public static void f1(int a) {
    System.out.println("int");
}
```

Работает, да. Но... ?

А такая перегрузка не работает

```
public static void f1(int a) {
    System.out.println("void");
}
public static int f1(int a) {
    System.out.println("int");
}
```

Почему?

Методы вызывают друг друга

- Просто внутри пишется имя метода с параметрами.
- А если метод внутри напишет себя...

Рекурсия

- Имеет серьезное математическое основание
  - рекурсивные функции – альтернатива Машине Тьюринга для задания функций
  - 3 базовых функции, 3 операции
- Со школы помним «рекуррентные соотношения».

Что надо уметь делать

- Описывать, в чем рекурсивность задачи

+

- Уметь явно указывать границу (когда рекурсия останавливается)

Пример #1. Натуральные числа

- Рекурсивное определение
  - 1 – натуральное число
  - Если x – натуральное, то x + 1 - натуральное

Пример #2. Факториал

- Факториал(0) = 1
- Факториал(n) = n * факториал(n-1)

Пример #3. Группа людей

- Рекурсивные определения имеют место и в реальном мире.

- Везде, где есть итерация, можно применит рекурсию.
  - Два человека – группа людей
  - Группа людей + человек – снова группа людей

Если не указать границу рекурсии
– что произойдет?

Stack Overflow

- Переполнение стека вызовов
  - «Довызывался»
- Максимальное значение можно настраивать в JVM (НО НЕ НУЖНО!)
  - Эксперименты давали 7000-8000.

Рекурсия и циклы

- Все циклы переписываются с помощью рекурсии!
  - Все-все. Но есть нюансы:
    - Цикл “while(true)” – это … ?
    - Еще нюанс?
- При этом она не должна проигрывать по сложности
  - См. плохой пример Фибоначчи
    - !

Цикл -> рекурсия

Поиск максимума в массиве

```
public static int maxOfArray(int[] array, int k) {
    if (k == array.length - 1) {
        return array[k];
    } else {
        int m = maxOfArray(array, k + 1);
        return m > array[k] ? m : array[k];
            // max(m, array[k])
    }
}

...
System.out.println(maxOfArray(array, 0));
```

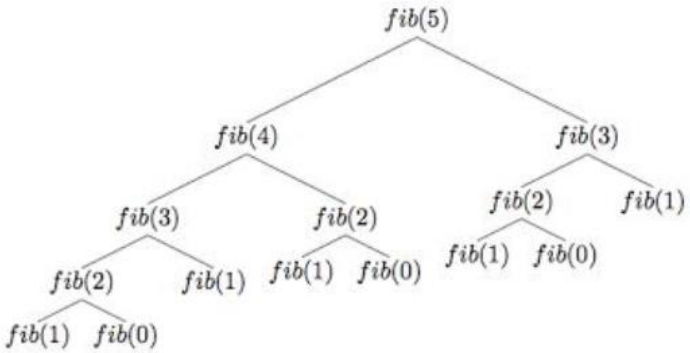
Кстати, это хвостовая рекурсия (после рек.вызова еще идут операции).
Не всегда это хорошо. Перепишете? Перепишите!

Плохая рекурсия

- Числа Фибоначчи

$$F_n = \begin{cases} 0, & n = 0; \\ 1, & n = 1; \\ F_{n-1} + F_{n-2}, & n \geq 2. \end{cases}$$

```
static ulong Fib(uint n)
{
    // Условие завершения.
    if (n < 2) return n;
    // Сведение к подзадаче и
    // рекурсивный вызов.
    return Fib(n - 1) + Fib(n - 2);
}
```



Слайд из лекции А.А.Марченко

Мемоизация

- Запоминать вычисленные на предыдущих шагах значения
- Использовать вычисленные значения вместо нахождения повторного решения
    - Фактически, «кэш»

Рекурсия с мемоизацией

```
static int Fib(int n)
{
    var cache = new int[n + 1];
    for(int i=0; i<=n;i++)
        if (i < 2) cache[i] = i;
        else cache[i] = -1;
    return FibRec(cache, n);
}
static int FibRec(int[]cache, int n)
{
    if (cache[n]>=0)
        return cache[n];
    cache[n]=FibRec(cache, n - 1) + FibRec(cache, n - 2);
    return cache[n];
}
```

Слайд из лекции А.А.Марченко