Программирование
1 семестр

ИТМО

Инструкции управления

Выражения и инструкции

ИТМО

- Выражения
  - Имеют значение и тип

```
(float)Math.cos(x) // float
a * (x + b) > 0  // boolean
isDone & noError // boolean
a*x*x + b*x + c   // double
```

- Инструкции
  - Описывают действие

```
System.out.println("!") ;
double y = a*x*x + b*x + c ;
result = Math.cos(x) ;
counter++ ;
```

Структурная парадигма

- Порядок исполнения инструкций
  - Последовательность
  - Ветвление
  - Цикл

- Инструкции исполняются последовательно одна за другой в том порядке, как они написаны в программе.

Структурная парадигма

ИТМО

- Порядок исполнения инструкций
  - Последовательность
  - Ветвление
  - Цикл

- Исполнение некоторых инструкций зависит от выполнения некоторого условия. Если условие истинно, исполняется один набор инструкций, если ложно - другой.

Структурная парадигма

ИТМО

- Порядок исполнения инструкций
  - Последовательность
  - Ветвление
  - Цикл

- Инструкции повторяются многократно, либо пока выполняется некоторое условие, либо некоторое количество раз, либо для всех элементов массива.

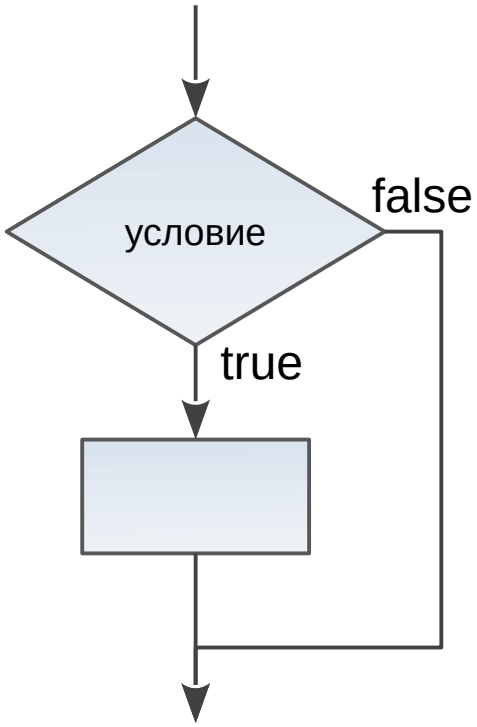
Блоки

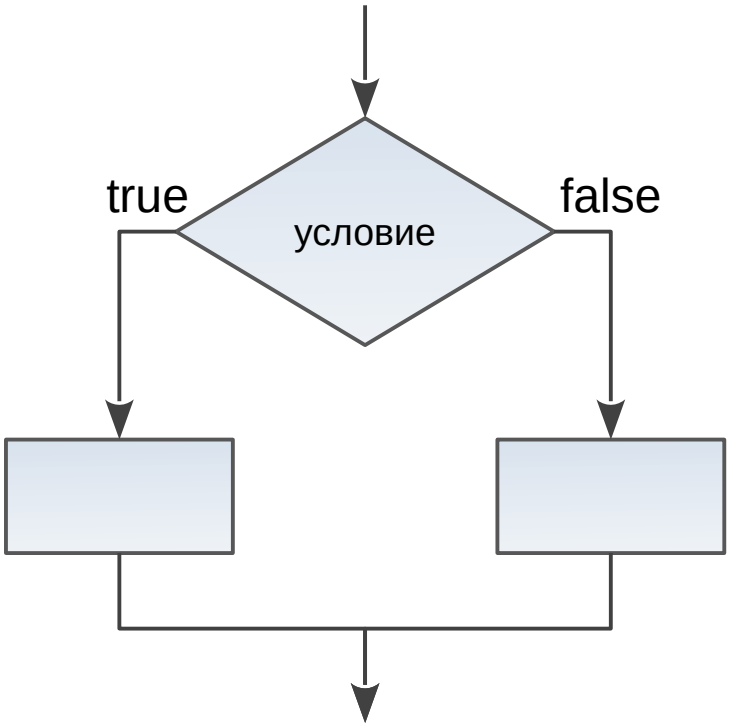
ИТМО

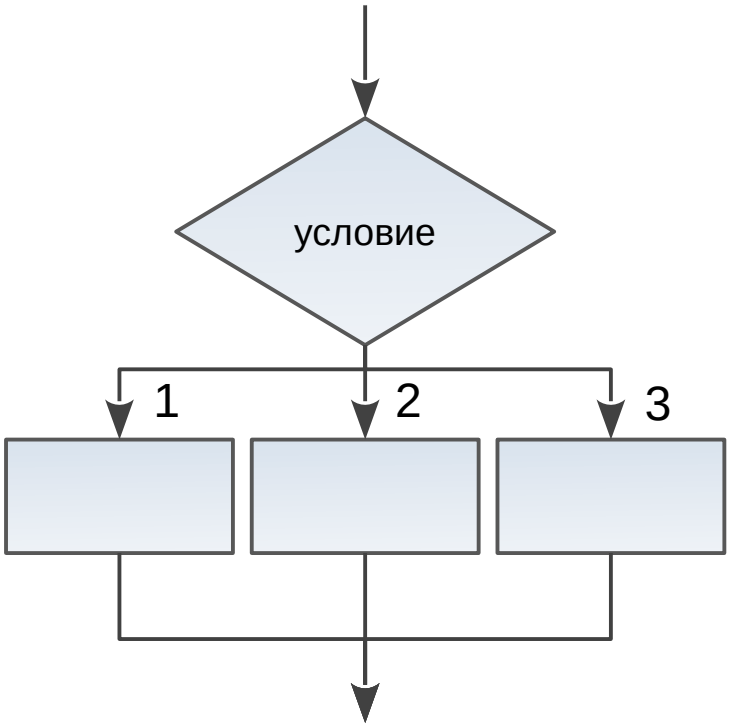
- Блок объединяет набор инструкций и ограничен фигурными скобками
- Локальные переменные, объявленные внутри блока, существуют до конца блока и не видны вне блока (variable scope)

```
int a = 2;
{                                    блок
  int a = 1;
  System.out.print(a); // 1
}
System.out.print(a); // 2
```

Ветвление

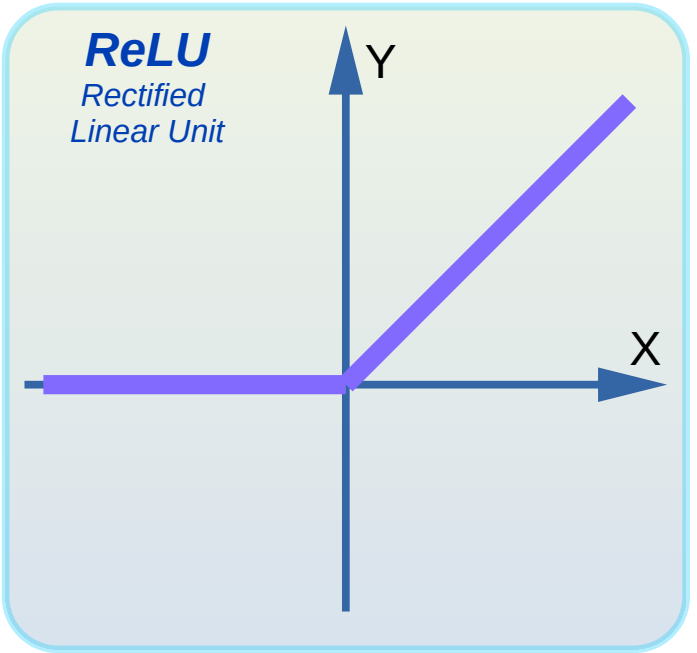






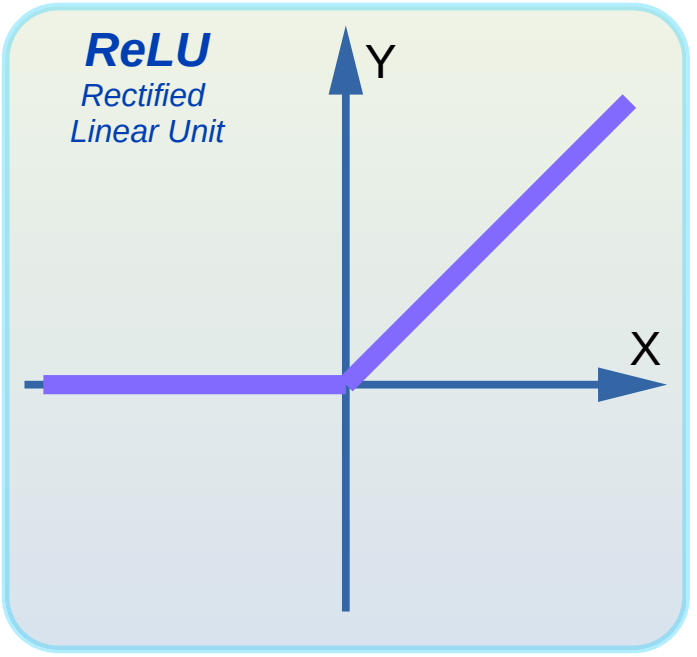
if

```
/* if (condition)
      true_statement;


*/
y = 0;
if (x > 0)
    y = x;
```



if-else

```
/* if (condition)
     true_statement;
   else
     false_statement;
*/

if (x > 0)
    y = x;
else
    y = 0;
```



Цепочки условий

```
if (x < 0)
  a = -1;
if (x == 0)
  a = 0;
if (x > 0)
  a = 1;
```

Цепочки условий

```
if (x < 0)
  a = -1;
if (x == 0)
  a = 0;
if (x > 0)
  a = 1;
```

```
if (x < 0)
  a = -1;
else if (x == 0)
  a = 0;
else
  a = 1;
```

Цепочки условий

```
if (x < 0)
  a = -1;
if (x == 0)
  a = 0;
if (x > 0)
  a = 1;
```

```
if (x < 0)
  a = -1;
else if (x == 0)
  a = 0;
else if (x > 0)
  a = 1;
System.out.print(a);
```

Цепочки условий

```
if (x < 0)
  a = -1;
if (x == 0)
  a = 0;
if (x > 0)
  a = 1;
```

```
if (x < 0)
  a = -1;
else if (x == 0)
  a = 0;
else if (x > 0)
  a = 1;
else
  System.out.print("Error!");
```

Вложенные условия

```
int x = 2; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  x = x + 1;
System.out.print(y/x);
// 2
```

$$\frac{|y|}{x}, \quad x \neq 0$$

$$\frac{y}{x+1}, \quad x = 0$$

Вложенные условия

```
int x = 2; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  x = x + 1;
System.out.print(y/x);
// 2
```

```
int x = 0; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  x = x + 1;
System.out.print(y/x);
// -4
```

Вложенные условия

ИТМО

```
int x = 0; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  x = x + 1;
System.out.print(y/x);
// Division by zero!
```

```
int x = 0; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  x = x + 1;
System.out.print(y/x);
// -4
```

Вложенные условия

```
int x = 0; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  System.out.print(x);
  x = x + 1;
System.out.print(y/x);
//
```

```
int x = 0; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  x = x + 1;
System.out.print(y/x);
// -4
```

Вложенные условия

```
int x = 0; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  System.out.print(x);
  x = x + 1;
System.out.print(y/x);
// -4
```

```
int x = 0; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  x = x + 1;
System.out.print(y/x);
// -4
```

Вложенные условия

ИТМО

```
int x = 0; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
else
  System.out.print(x);
  x = x + 1;
System.out.print(y/x);
// -4
```

```
int x = 0; int y = -4;
if (x != 0)
  if (y < 0)
    y = -y;
  else
    System.out.print(x);
x = x + 1;
System.out.print(y/x);
// -4
```

Вложенные условия

ИТМО

```
int x = 0; int y = -4;
if (x != 0) {
   if (y < 0) {
      y = -y;
   }
} else {
   x = x + 1;
}
System.out.print(y/x);
```

Тип условия - boolean

```
int x = 2
int product = 1;


if (x != 0) {
  product *= x;
}
```

```
int x = 2
int product = 1;


if (x) {
  product *= x;
}
```

Переменная типа boolean

```
int x = 2
int product = 1;
boolean isNotZero = x != 0;

if (isNotZero) {
  product *= x;
}
```

```
int x = 2
int product = 1;


if (x) {
  product *= x;
}
```

Переменная типа boolean

ИТМО

```
int x = 2
int product = 1;
boolean isNotZero = x != 0;

if (isNotZero) {
  product *= x;
}
```

```
int x = 2
int product = 1;
boolean isNotZero = x != 0;

if (isNotZero == true) {
  product *= x;
}
```

Не нужно усложнять

ИТМО

```
boolean isZero;
boolean isNotZero = x != 0;
```

```
boolean isZero;
boolean isNotZero = x != 0;

if (isNotZero) {
   isZero = false;
} else {
   isZero = true;
}
```

Не нужно усложнять

ИТМО

```
boolean isZero;
boolean isNotZero = x != 0;

isZero = ! isNotZero
isZero = ! x != 0;
isZero = x == 0;
```

```
boolean isZero;
boolean isNotZero = x != 0;

if (isNotZero) {
   isZero = false;
} else {
   isZero = true;
}
```

Логические операторы для boolean и целых

ИТМО

```
boolean a = false;
boolean b = true;
// логические операции
notOp = ! a;
andOp = a & b;
orOp = a | b;
xorOp = a ^ b;
```

```
int a = 0b0011;
int b = 0b0101;
// побитовые операции
notOp = ~ a;    // 1100
andOp = a & b;  // 0001
orOp = a | b;   // 0111
xorOp = a ^ b;  // 0110
```

Стандартные логические операторы

```
int x = 1;
int y = 5;

      true          true
if (x != 0 & y/x > 0) {
   ...
}
```

- Вычисляются оба операнда
- Потом выполнятся логическое И

Стандартные логические операторы

ИТМО

```
int x = 0;
int y = 5;

      false     error
if (x != 0 & y/x > 0) {
   ...
}
```

- Вычисляются оба операнда
- Потом выполнятся логическое И
- y/x - Ошибка!

Сокращенные логические операторы

```
int x = 0;
int y = 5;

      false          не вычисляем
if (x != 0 && y/x > 0) {
   ...
}
```

- Вычисляется левая часть
- Если false, то результат точно будет false
- Правая часть вычисляется только, если левая == true

Сокращенные логические операторы

ИТМО

```
false & false = false
false & true  = false
true  & false = false
true  & true  = true
```

```
false | false = false
false | true  = true
true  | false = true
true  | true  = true
```

```
false && any   = false

true  && false = false
true  && true  = true
```

```
false || false = false
false || true  = true
true  || any   = true
```

Условный оператор

```
if (x < 0) {
   y = 0;
} else {
   y = x;
}
```

- Если в блоках if и else выполняется присваивание одной и той же переменной

Условный оператор

```
if (x < 0) {
   y = 0;
} else {
   y = x;
}
```

```
condition ? value_if_true : value_if_false
```

```
y = x < 0 ? 0 : x;
```

- Если в блоках if и else выполняется присваивание одной и той же переменной
- Можно использовать условный оператор - он короче!

Условный оператор (вложенный)

ИТМО

```
if (x < 0) {
   y = 0;
} else {
   y = x;
}
```

condition ? *value_if_true* : *value_if_false*

```
y = x < 0 ? 0 : x;
```

```
if (x < 0) {
   a = -1;
} else if (x > 0) {
   a = 1;
} else {
   a = 0;
}
```

```
a = x < 0 ? -1 : (x > 0 ? 1 : 0);
```

Условный оператор - пример

ИТМО

```
int rub = 292;
int cop = 41;
                    10...19      1234567890
String rubStr = rub/10%10 == 1 || (rub+9)%10 >= 4 ? " рублей" :
                rub%10 > 1 ? " рубля" : " рубль";
String copStr = cop/10%10 == 1 || (cop+9)%10 >= 4 ? " копеек" :
                cop%10 > 1 ? " копейки" : " копейка";


System.out.print(rub + rubStr +" "+ cop + copStr);
// 292 рубля 41 копейка
```

Множественный выбор

ИТМО

```
if (month == 2) {
  days = isLeapYear? 29 : 28;
} else if (month == 4 ||
           month == 6 ||
           month == 9 ||
           month == 11) {
  days = 30;
} else {
  days = 31;
}
```

- Запутанный код
- Много сравнений

Множественный выбор - инструкция switch

ИТМО

```
switch (month) {
  case 2:
   days = isLeapYear? 29 : 28;
   break;
  case 4: case 6:
  case 9: case 11:
   days = 30; break;
  default: days = 31; break;
}
```

- Менее запутанный код
- Выражение вычисляется один раз, потом переход на нужный case или default (хотя компилятор может иногда создать цепочку if)
- Далее код выполняется, пока не встретится break

Инструкция switch - так можно, но не нужно

ИТМО

```
int days = 31;
switch (month) {
  case 2:
    days -= isLeapYear? 1 : 2;
  case 4:
  case 6:
  case 9:
  case 11: days -= 1;
}
```

- Если break отсутствует, код продолжает выполняться
- Иногда этим удобно воспользоваться
- В большинстве случаев так делать не нужно
- Можно все сломать

Инструкция switch - так лучше

ИТМО

```
int days;
switch (month) {
  case 2:
    days = isLeapYear? 29 : 28;
    break;
  case 4, 6, 9, 11:
    days = 30; break;
  case 1, 3, 5, 7, 8, 10, 12:
    days = 31; break;
  default: days = 0; break;
}
```

- Можно перечислить значения в case через запятую
- default использовать не обязательно, но так меньше потенциальных ошибок

Инструкция switch - так короче

ИТМО

```
int days;
switch (month) {
   case 2 ->
     days = isLeapYear? 29 : 28;
   case 4,6,9,11 ->
     days = 30;
   case 1,3,5,7,8,10,12 ->
     days = 31;
   default -> days = 0;
}
```

- Новый синтаксис, вместо двоеточия - стрелка
- break не используется, после каждого case - выход из блока switch
- default почти обязателен
- Должны быть обработаны ВСЕ возможные значения

Выражение switch - еще удобнее

ИТМО

```
int days = switch (month) {
   case 2 -> isLeapYear? 29:28;
   case 4,6,9,11 -> 30;
   case 1,3,5,7,8,10,12 -> 31;
   default -> 0; // wrong month
}
```

```
switch (expression) {
   case value -> result;
   case value,value -> result;
   default -> result;
}
```

- Если во всех ветках одно и то же присваивание
- После стрелки оставляем только значение

Выражение switch (еще вариант)

```
int days = switch (month) {
  case 2 -> {
    if (year%400 == 0) { yield 28; }
    else if (year%4 == 0 && year%100 != 0) { yield 29; }
    else { yield 28; }
  }
  case 4, 6, 9, 11 -> 30;
  case 1, 3, 5, 7, 8, 10, 12 -> 31;
  default -> 0; // wrong month
}
```

- Если для вычисления значения требуется более сложный код, который не помещается в одно выражение
- Блок case - в фигурных скобках
- Возврат значения - с помощью инструкции **yield**

Массивы

ИТМО

- Массив - ссылочный тип (не примитивный)
- Может иметь ноль или больше элементов
- Характеристики массива (фиксируются при создании):
  - расположение в памяти начального элемента
  - длина (количество элементов)
  - тип (и размер в байтах) элементов
- Характеристики элементов массива:
  - индекс (числовой последовательный уникальный)

Работа с массивом

ИТМО

- Массив
  - объявление
  - создание
  - инициализация
  - тоже инициализация
  - присваивание значения
  - получение элемента
  - длина массива

```
type[] name = new type[len];
   int[] array; // null
   array = new int[3];
   int[] x = new int[2]{2,1};
   int[] y = {3, 4, 5};
                [0] [1] [2]
   y[1] = -10;
   System.out.print(y[0]);
   int len = x.length; // 3
```

Ошибки при работе с массивом

```
int[] array; // array = null
System.out.print(array[0]);
System.out.print(array.length);

array = new int[4];
System.out.print(array[4]);
System.out.print(array[-4]);
```

- NullPointerException (NPE)
  - массив еще не создан
  - ссылка = null
- ArrayIndexOutOfBounds Exception
  - выход индекса за границы массива

Переменные в памяти (локальные)

ИТМО

- Память в Java - динамическое управление
  - выделяется автоматически
  - освобождается автоматически
  - 2 основные области: стек и куча

- Стек (Stack)
  - локальные переменные
  - параметры метода
  - примитивные значения и ссылки

- Куча (Heap)
  - объекты (на которые указывают ссылки)
    - массивы, строки, ...
  - чистится сборщиком мусора

Примитивные и ссылочные типы

ИТМО

- Примитивные (значение)
  - Данные - одиночное значение
  - Значение хранится в стеке
  - У значения фиксированный размер в байтах
  - Копируется значение
  - Сравнивается значение
- Ссылочные (объект + ссылка)
  - Данные - составной объект, который хранится в куче
  - Ссылка на адрес объекта в памяти хранится в стеке
  - У ссылки фиксированный размер - 4 или 8 байт
  - Копируется ссылка
  - Сравнивается ссылка

Примитивы и одномерные массивы в памяти

ИТМО

```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```

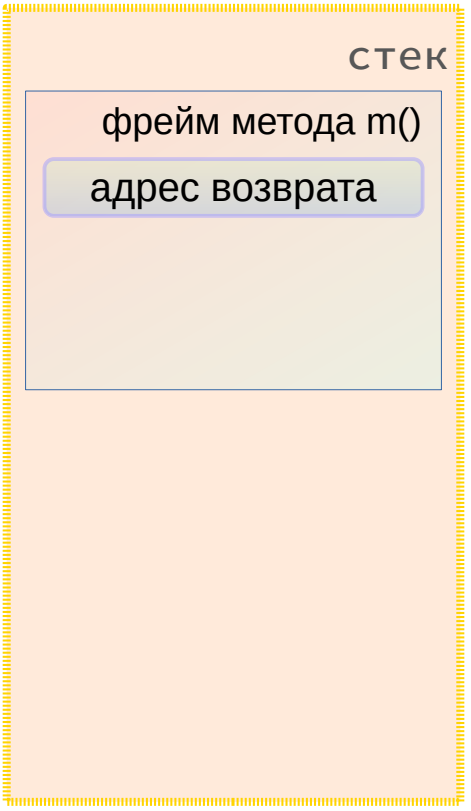
стек

куча

Примитивы и одномерные массивы в памяти

ИТМО

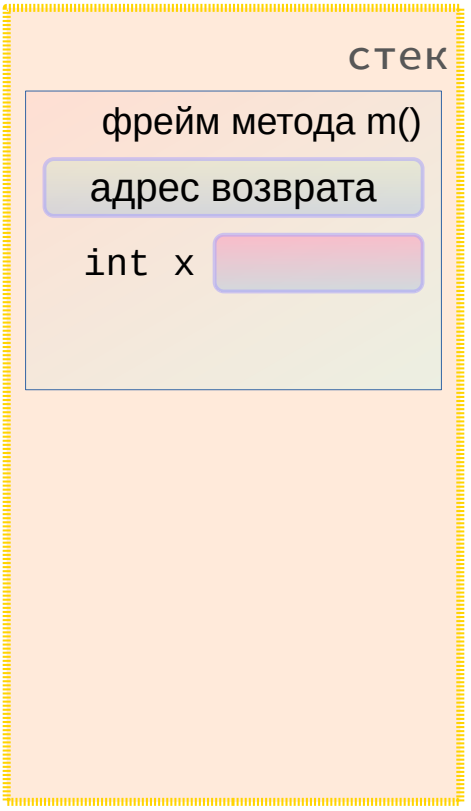
```
void m() {
   int x;
   int[] y;
   x = 1;
   y = new int[3];
   y[1] = 5;
   y[0] = x;
   x = y.length;
   y = new int[]{3,4};
}
```



Примитивы и одномерные массивы в памяти

ИТМО

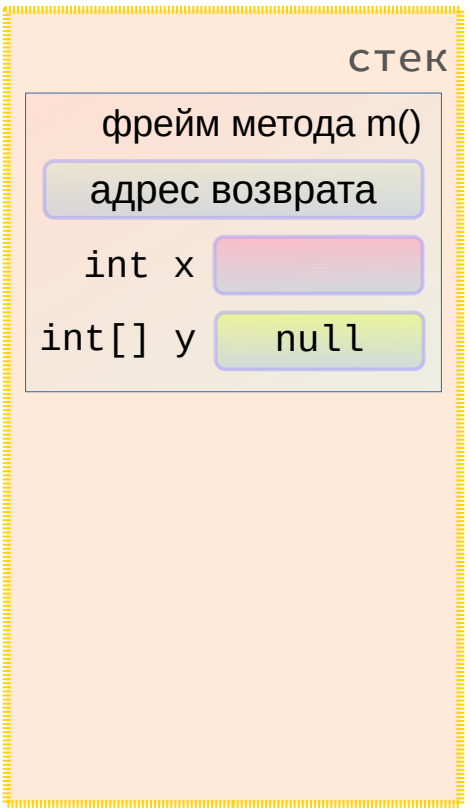
```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```

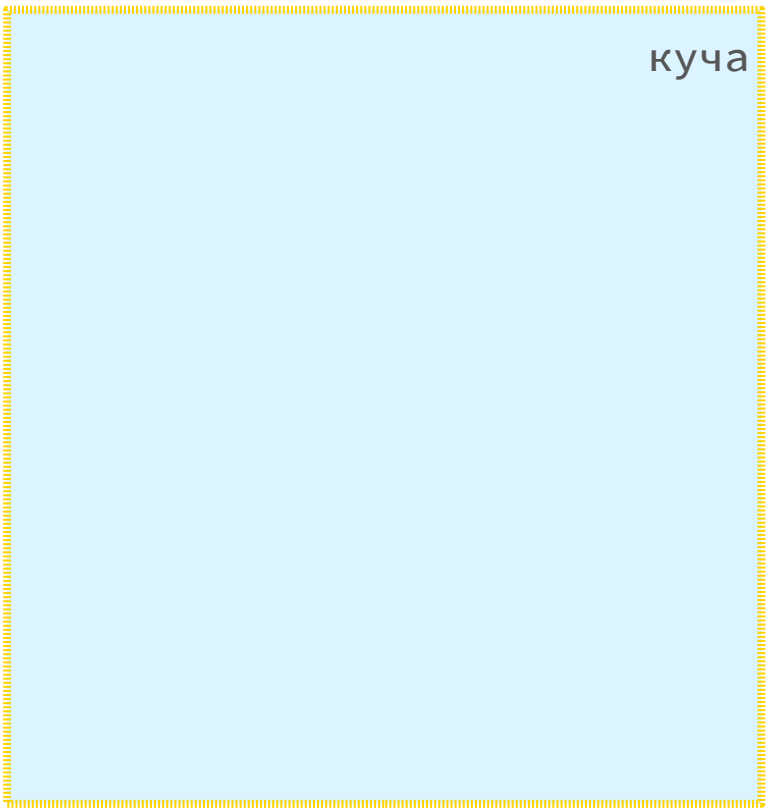


Примитивы и одномерные массивы в памяти

ИТМО

```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```

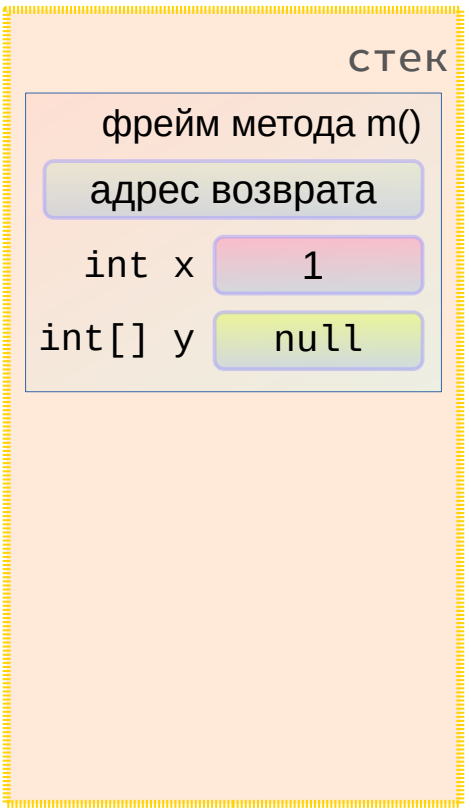


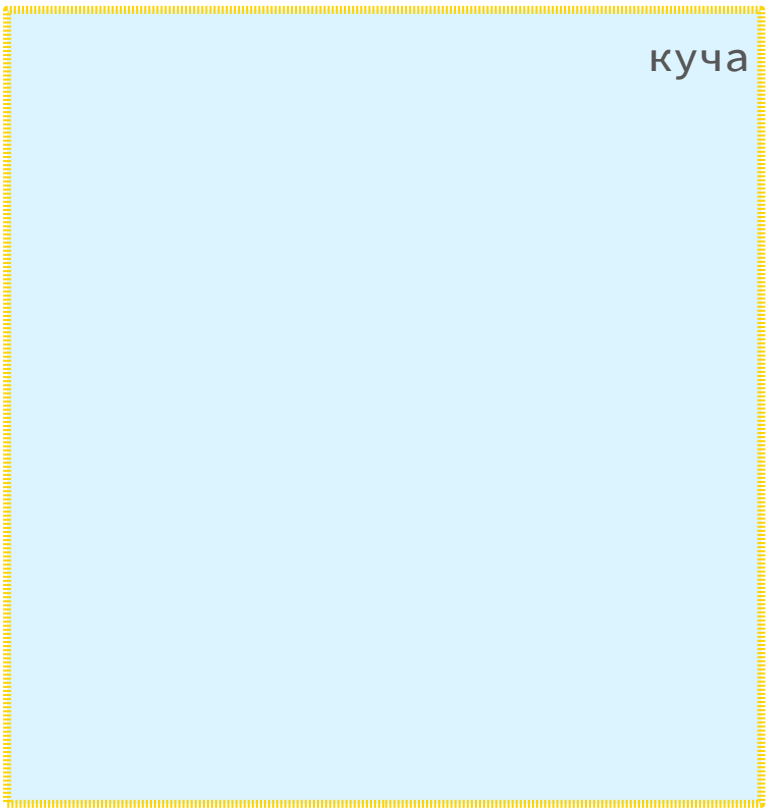


Примитивы и одномерные массивы в памяти

ИТМО

```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```

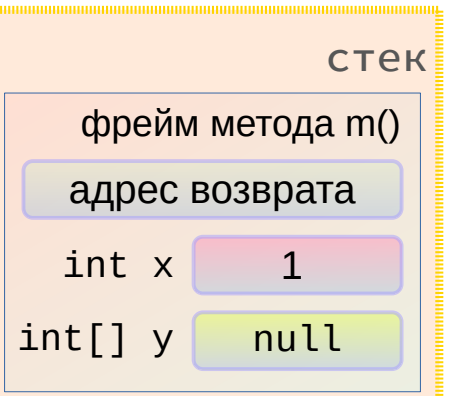


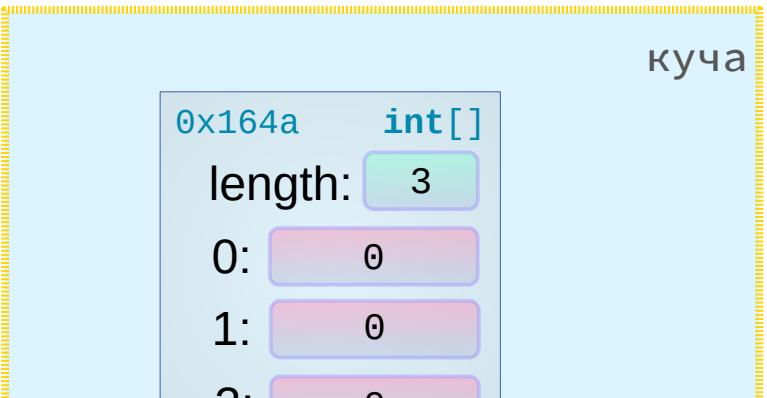


Примитивы и одномерные массивы в памяти

ИТМО

```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```

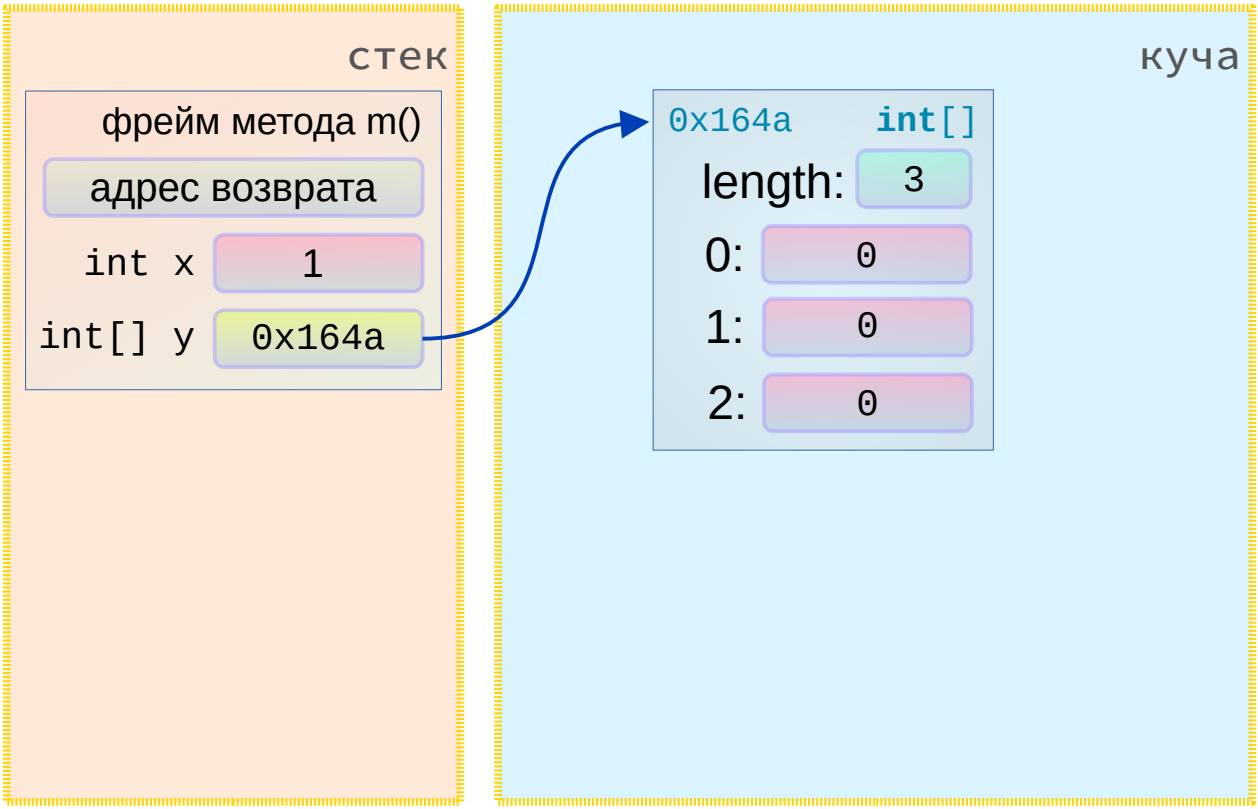




Примитивы и одномерные массивы в памяти

ИТМО

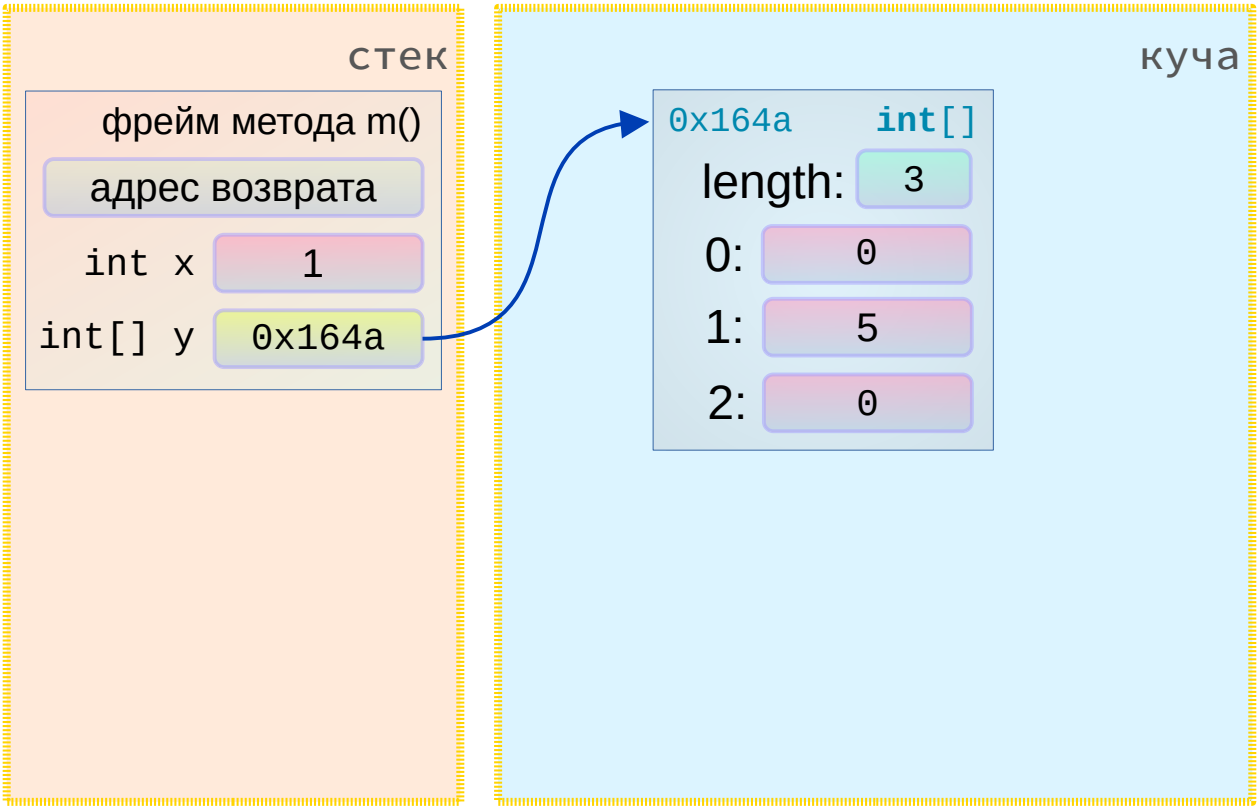
```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```



Примитивы и одномерные массивы в памяти

ИТМО

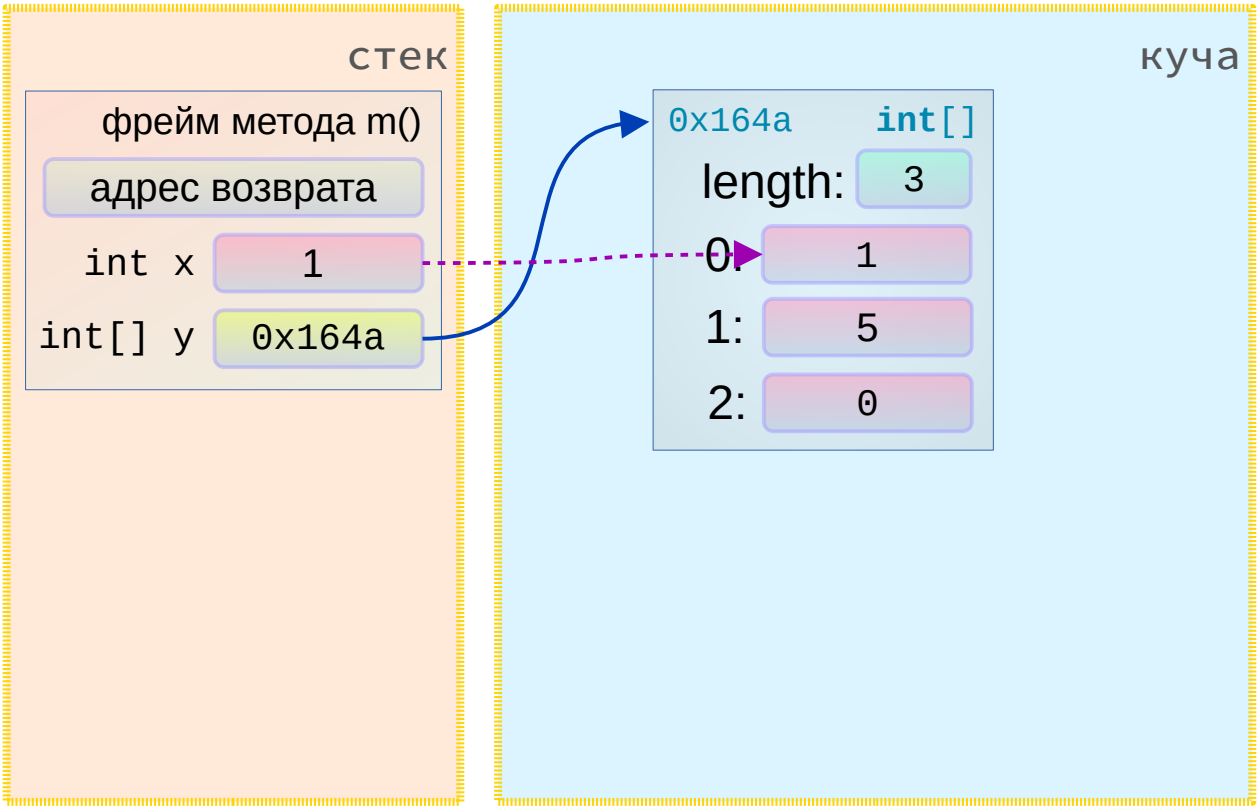
```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```



Примитивы и одномерные массивы в памяти

ИТМО

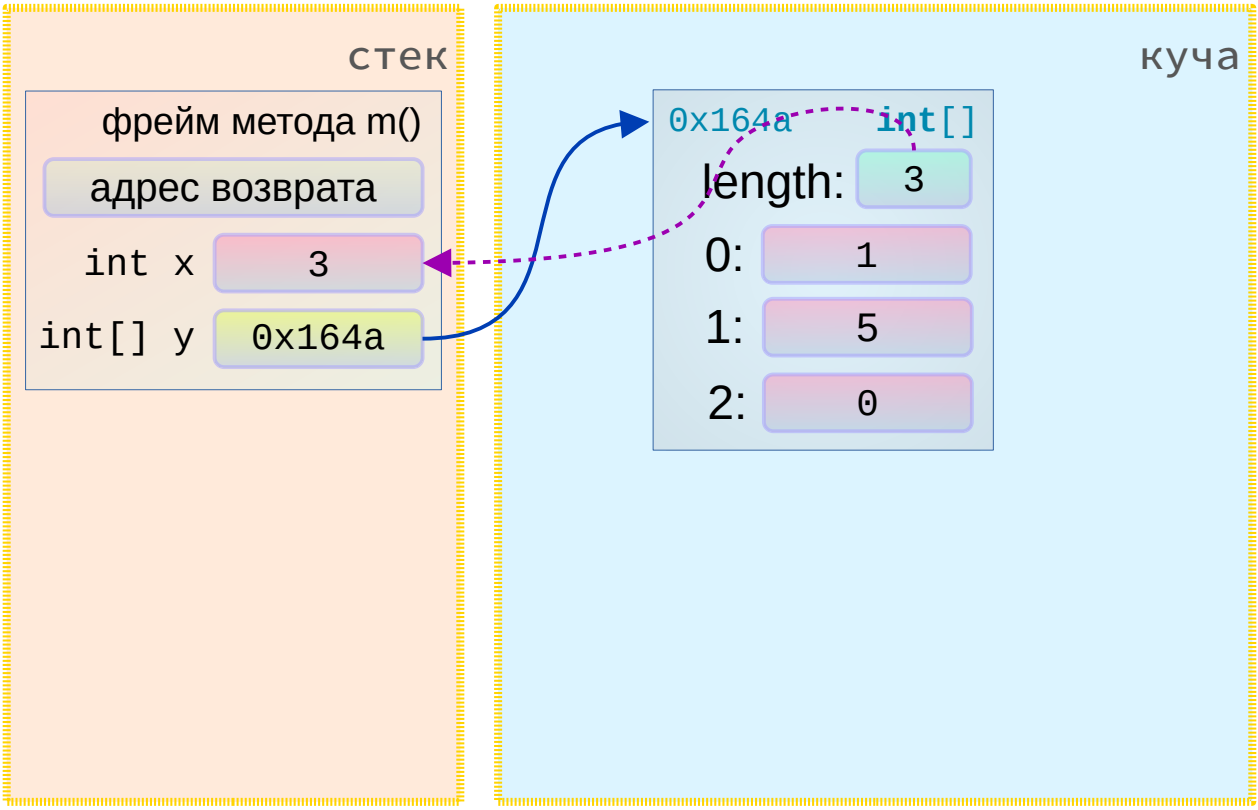
```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```



Примитивы и одномерные массивы в памяти

ИТМО

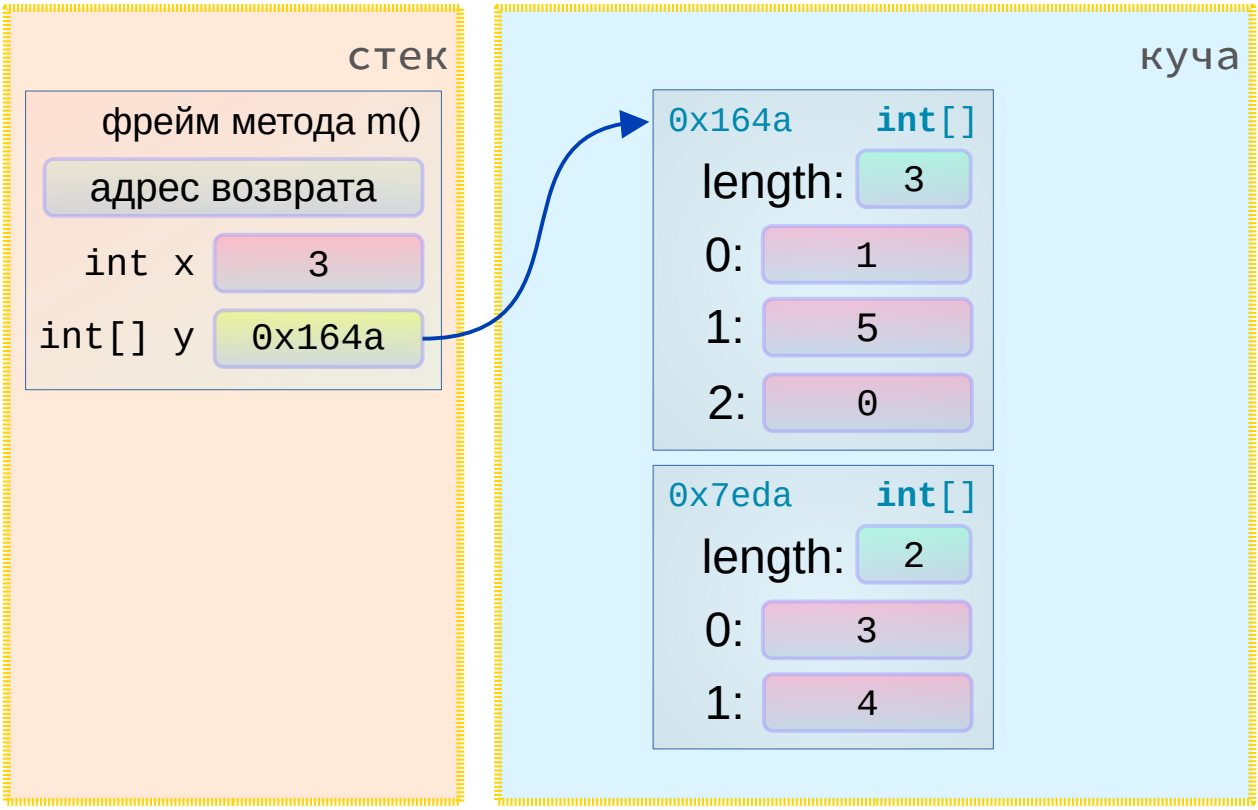
```
void m() {
   int x;
   int[] y;
   x = 1;
   y = new int[3];
   y[1] = 5;
   y[0] = x;
   x = y.length;
   y = new int[]{3,4};
}
```



Примитивы и одномерные массивы в памяти

ИТМО

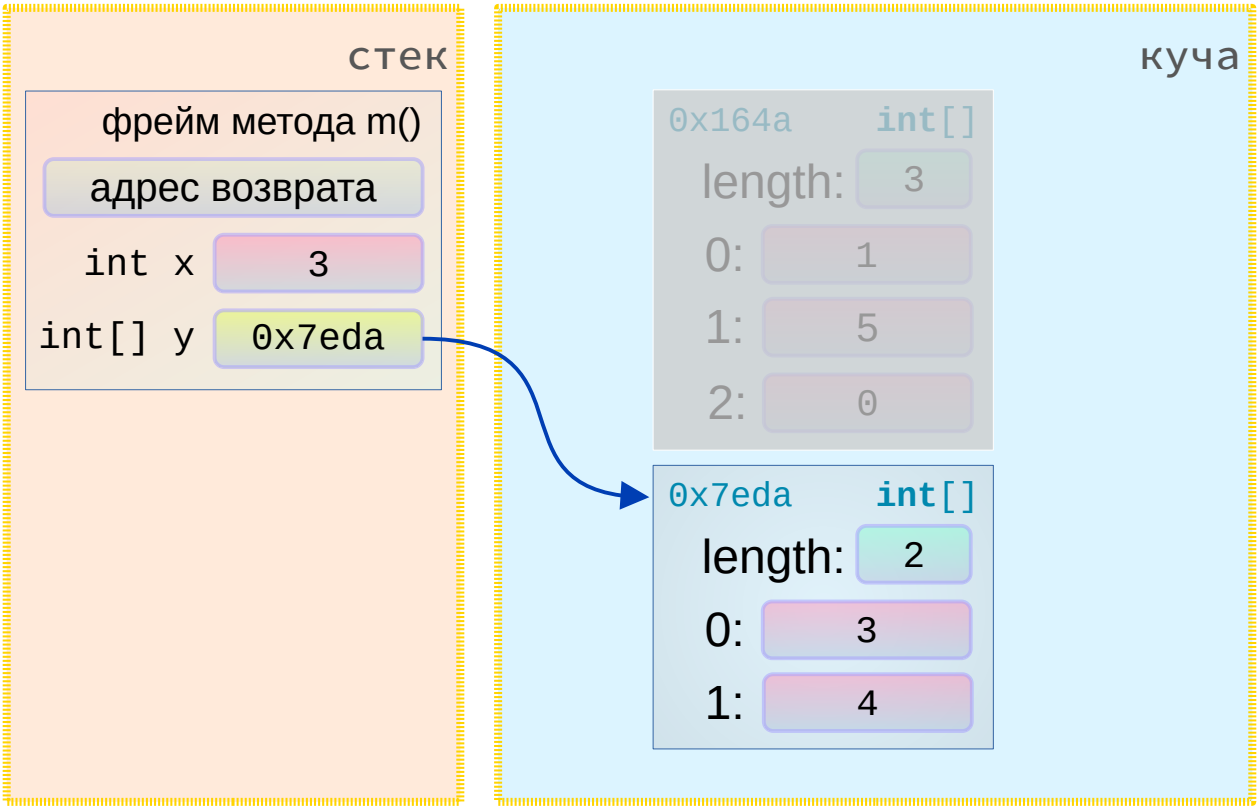
```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```



Примитивы и одномерные массивы в памяти

ИТМО

```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```



Примитивы и одномерные массивы в памяти

ИТМО

```
void m() {
  int x;
  int[] y;
  x = 1;
  y = new int[3];
  y[1] = 5;
  y[0] = x;
  x = y.length;
  y = new int[]{3,4};
}
```

стек

адрес возврата

куча

0x164a int[]

length: 3

0: 1

1: 5

2: 0

0x7eda int[]

length: 2

0: 3

1: 4

Многомерные массивы

- 1 способ - плотный (dense) или регулярный (regular) массив
  - Единый блок хранения в памяти
  - Высокая производительность
  - Возможность гибко менять форму и размерность
- 2 способ - массив массивов меньшей размерности
  - Может храниться частями
  - Производительность ниже, но зависит от реальной формы
  - Возможность иметь не только прямоугольную форму

Многомерные массивы

ИТМО

- Java - массив массивов
  - Массив со ссылками на другие массивы
  - Можно инициализировать каждый уровень отдельно
  - Можно инициализировать сразу весь массив

```
int[][][] D3Array = {{{1,2},{3,4}},{5,6},{7,8}}};
int[][] D2Array = new int[3][]; // {null,null,null}
D2Array[0] = {10,20,30,40};
int[][] squareArray = new int[5][5]; // filled with 0's
```

Двумерные массивы в памяти

ИТМО

```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```

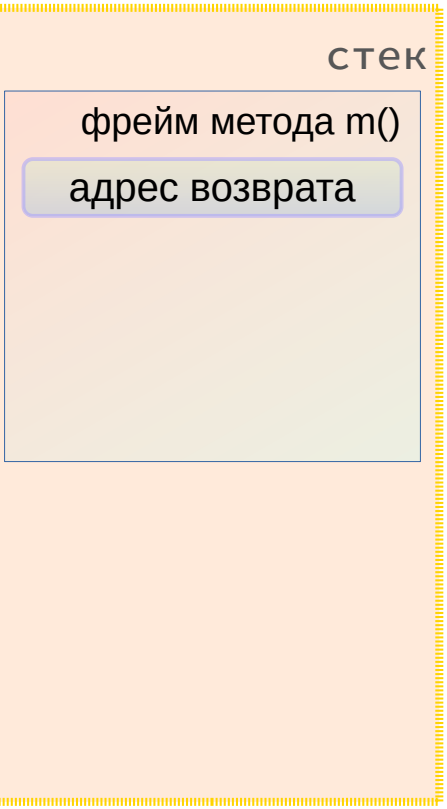
стек

куча

Двумерные массивы в памяти

ИТМО

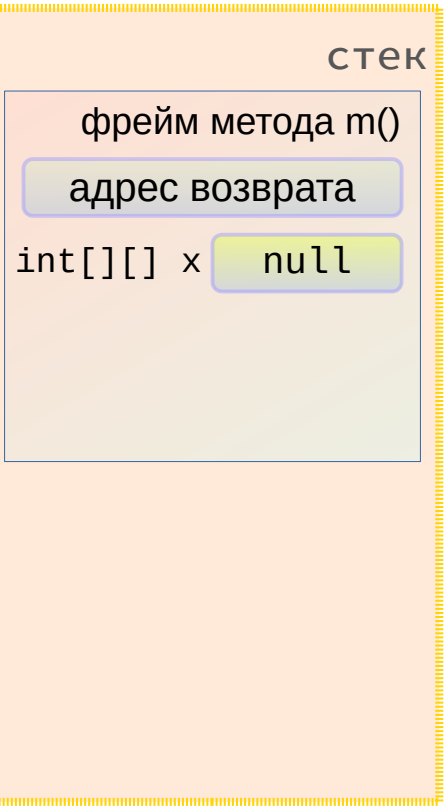
```
void m() {
   int[][] x;
   int[][] y;
   int[] z = {2,3,5};
   x = new int[2][2];
   y = x;
   x = new int[2][];
   x[0] = z;
   y[1] = new int[]{1};
   z[2] = 4;
}
```



Двумерные массивы в памяти

ИТМО

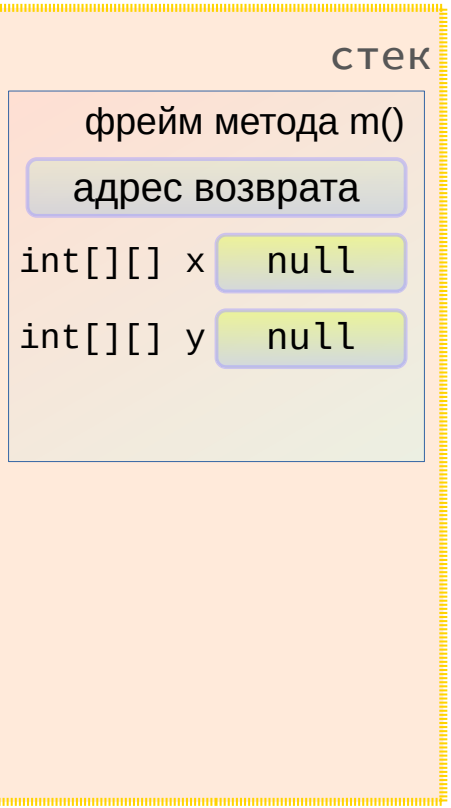
```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



Двумерные массивы в памяти

ИТМО

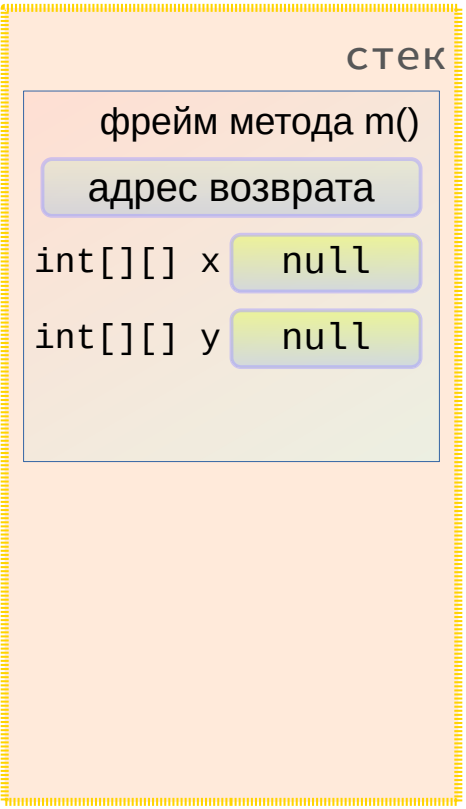
```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```

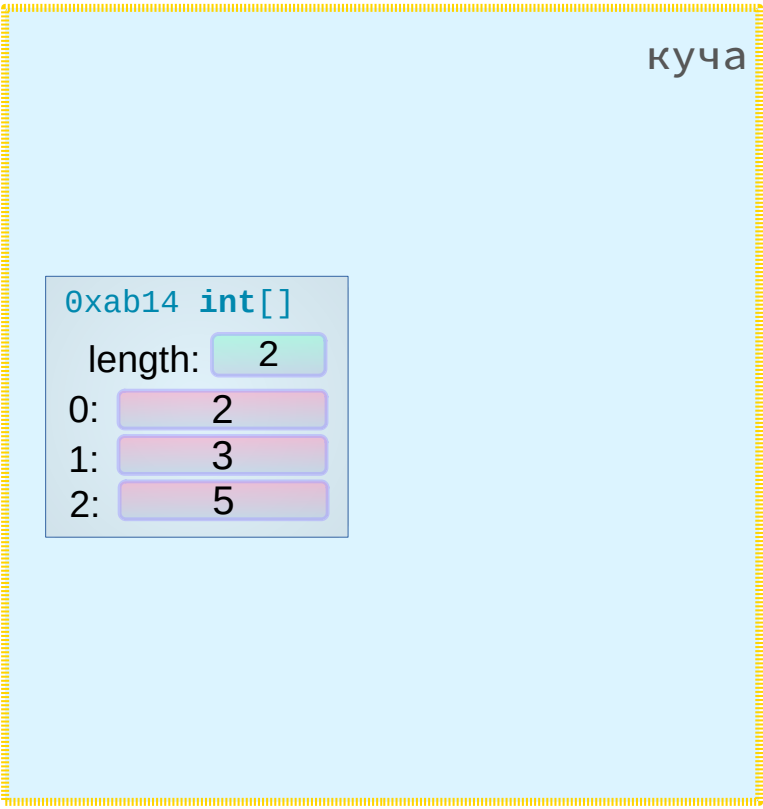


Двумерные массивы в памяти

ИТМО

```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```

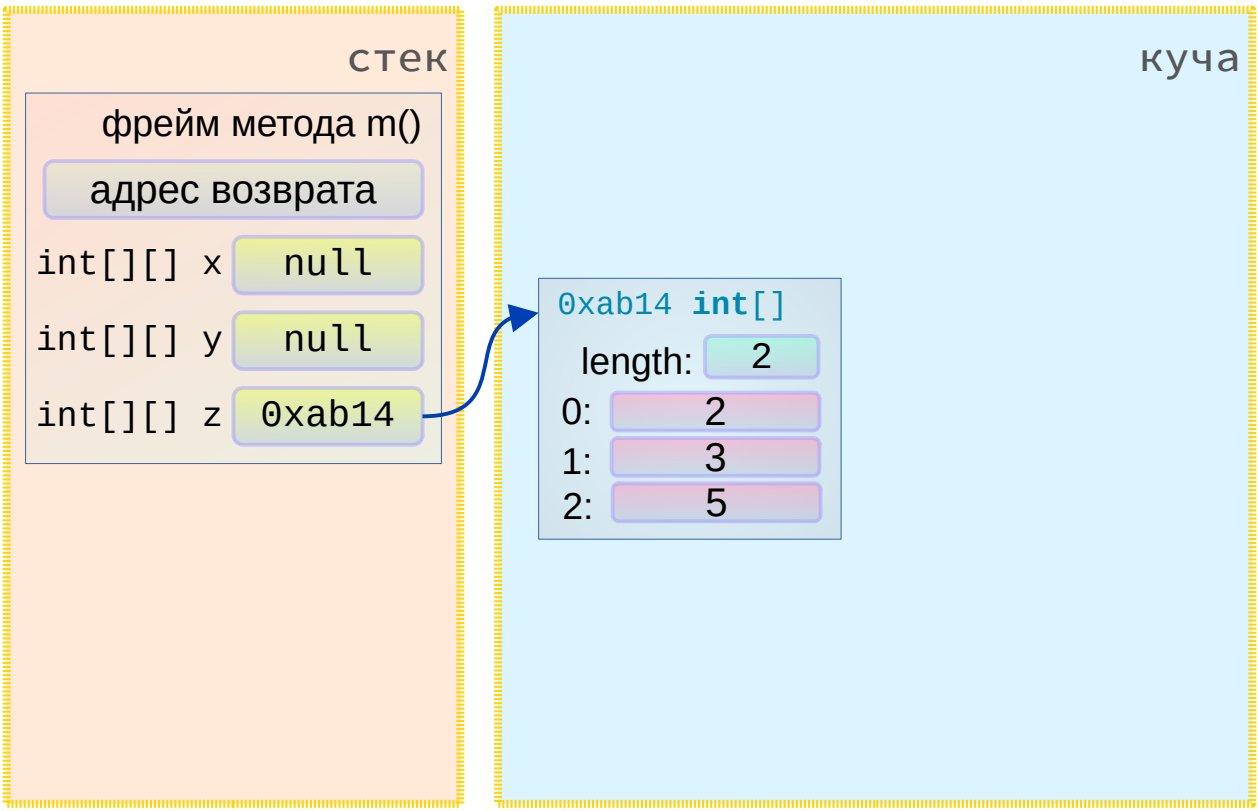




Двумерные массивы в памяти

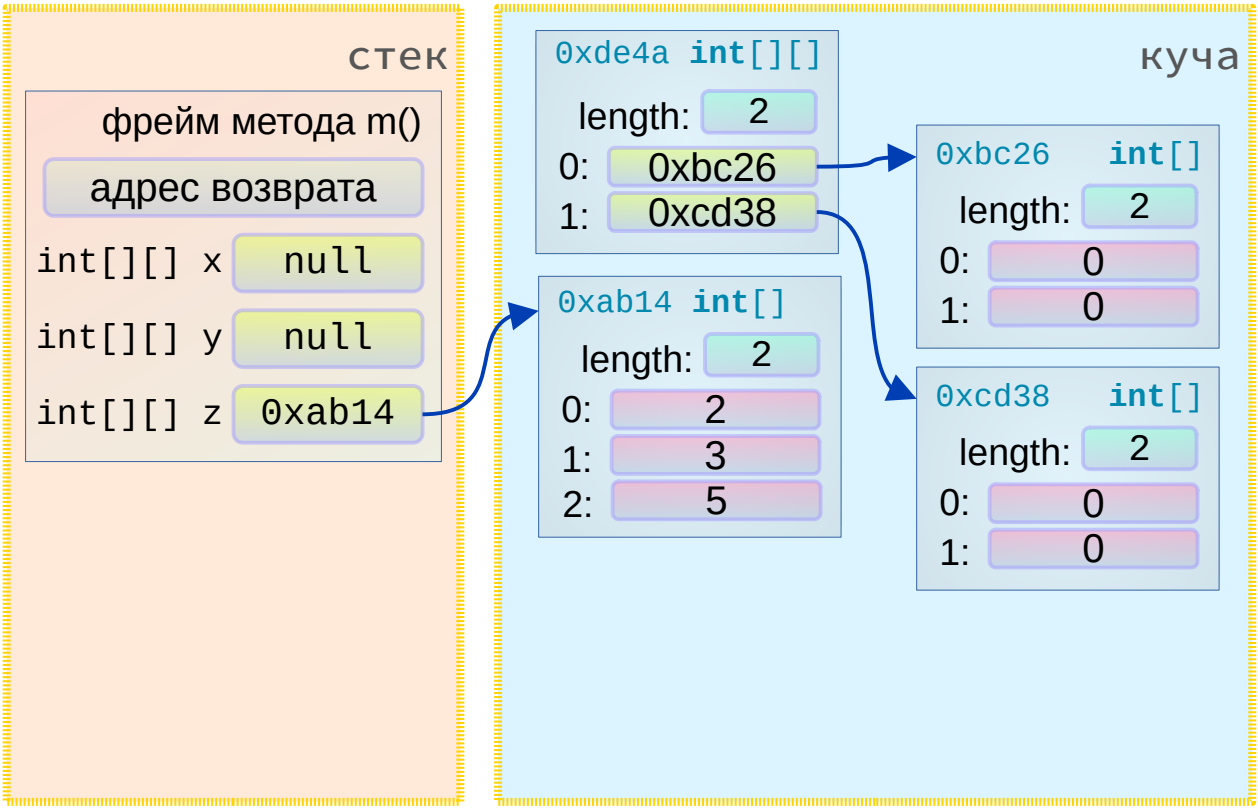
ИТМО

```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



Двумерные массивы в памяти

```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



Множественный выбор - инструкция switch

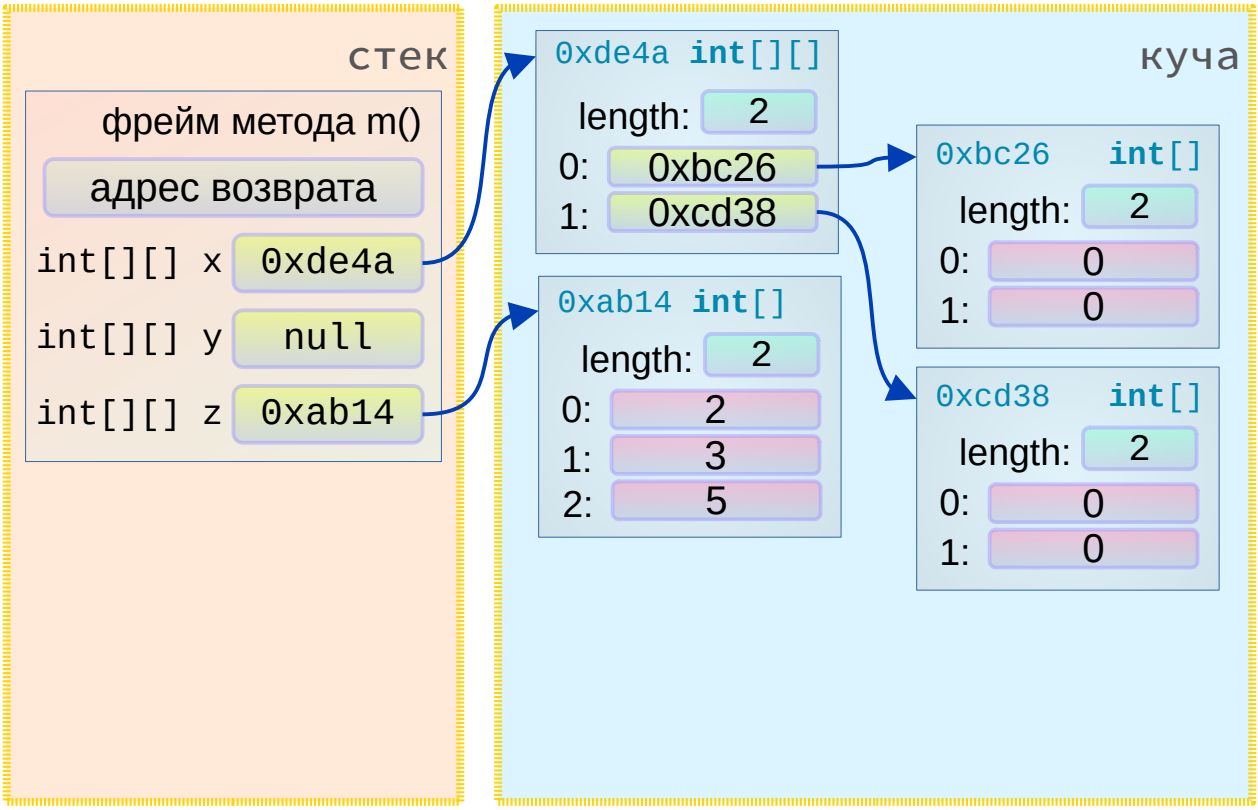
ИТМО

```
switch (expression) {
  case value:
    [ statements; ] [ break; ]
  case value,value:
    [ statements; ] [ break; ]
  [ default: ]
    [ statements; ] [ break; ]
}
```

- Возможные типы данных для *expression*:
  - int, byte, short, char
  - обертки для этих типов
  - String
  - enum
- Нельзя использовать:
  - long, boolean, float, double

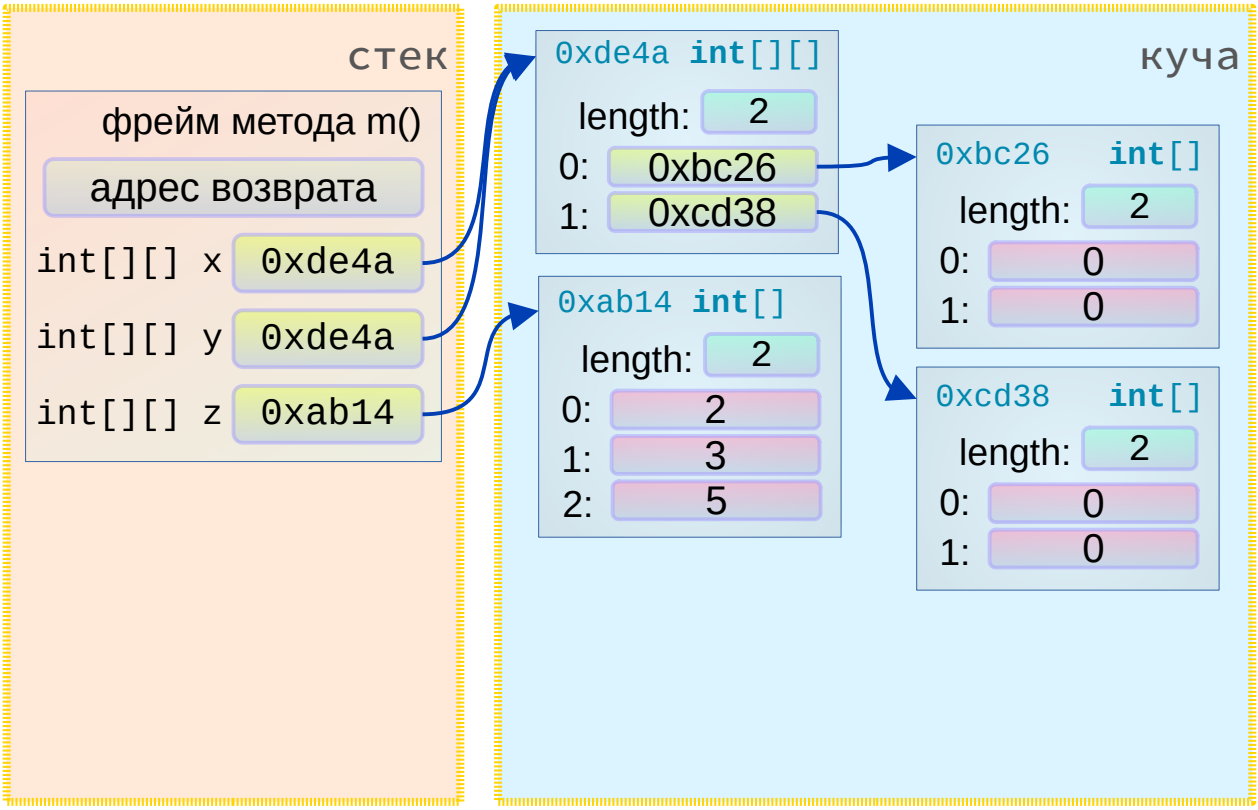
Двумерные массивы в памяти

```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



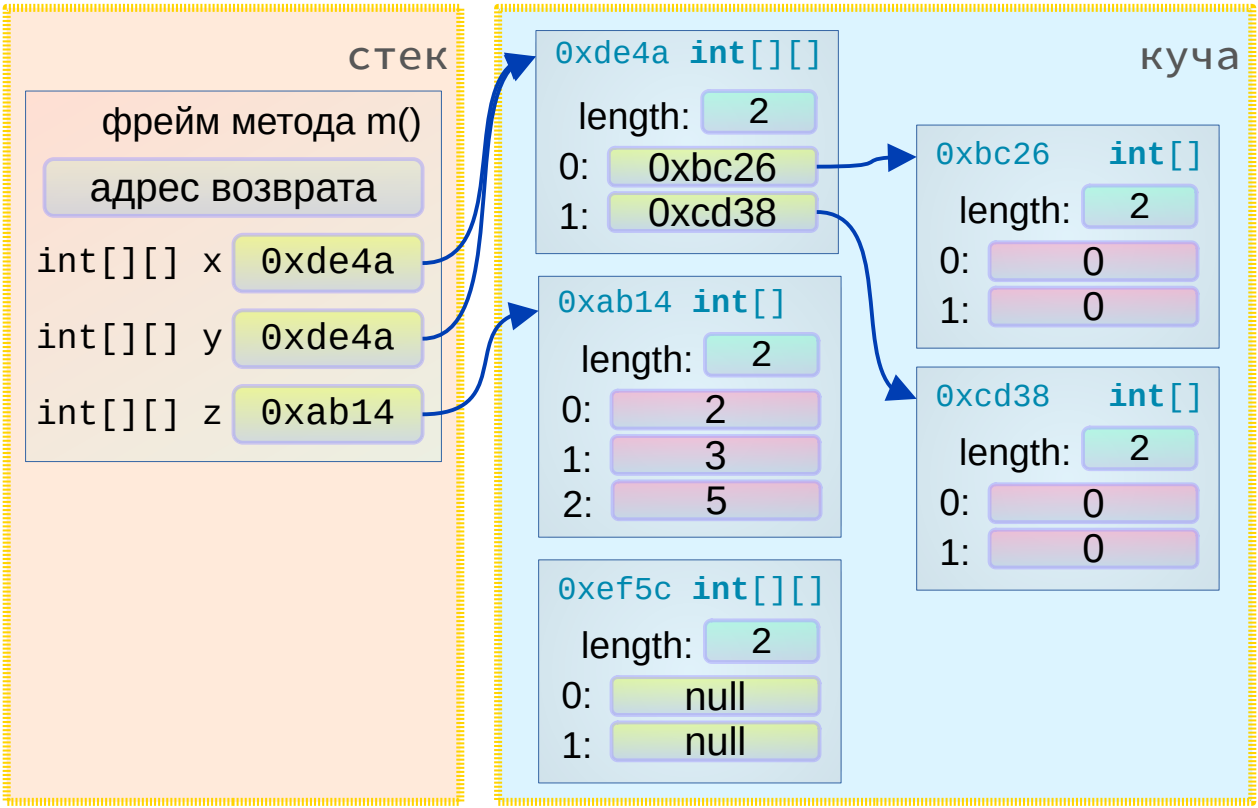
Двумерные массивы в памяти

```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



Двумерные массивы в памяти

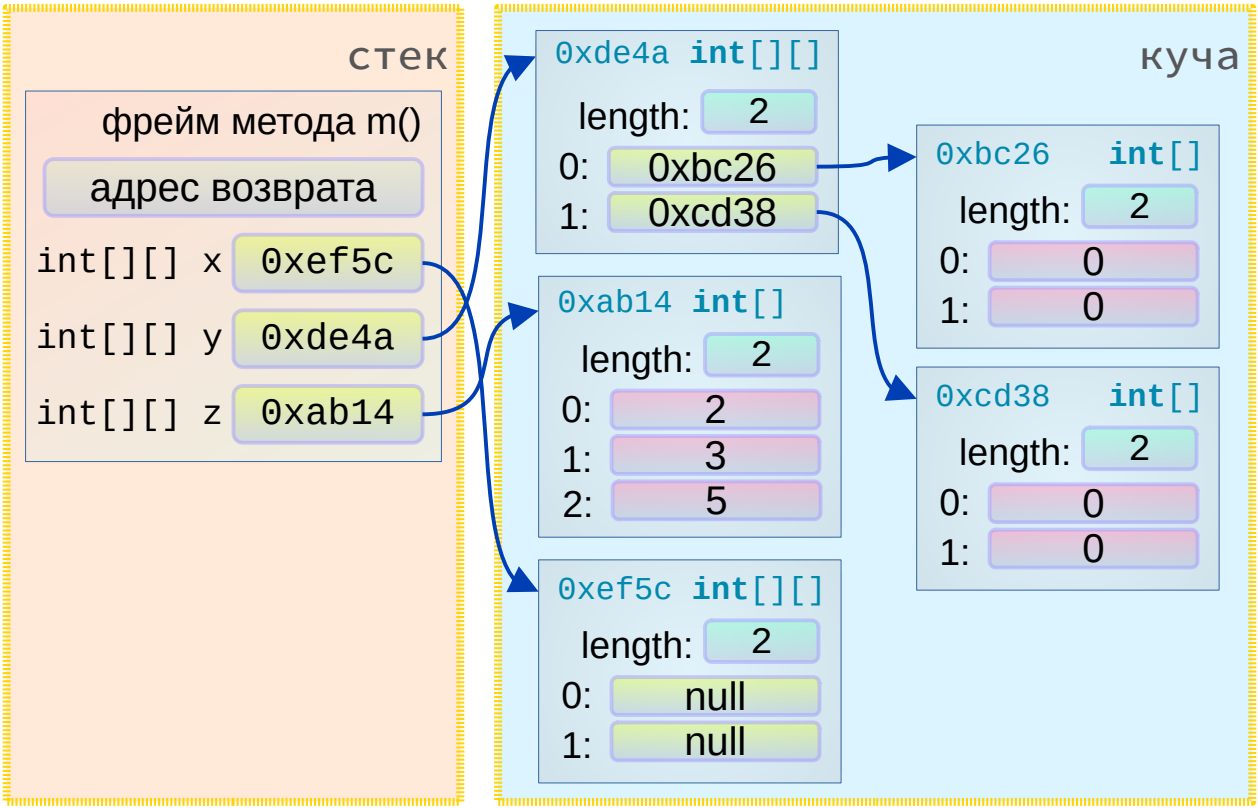
```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



Двумерные массивы в памяти

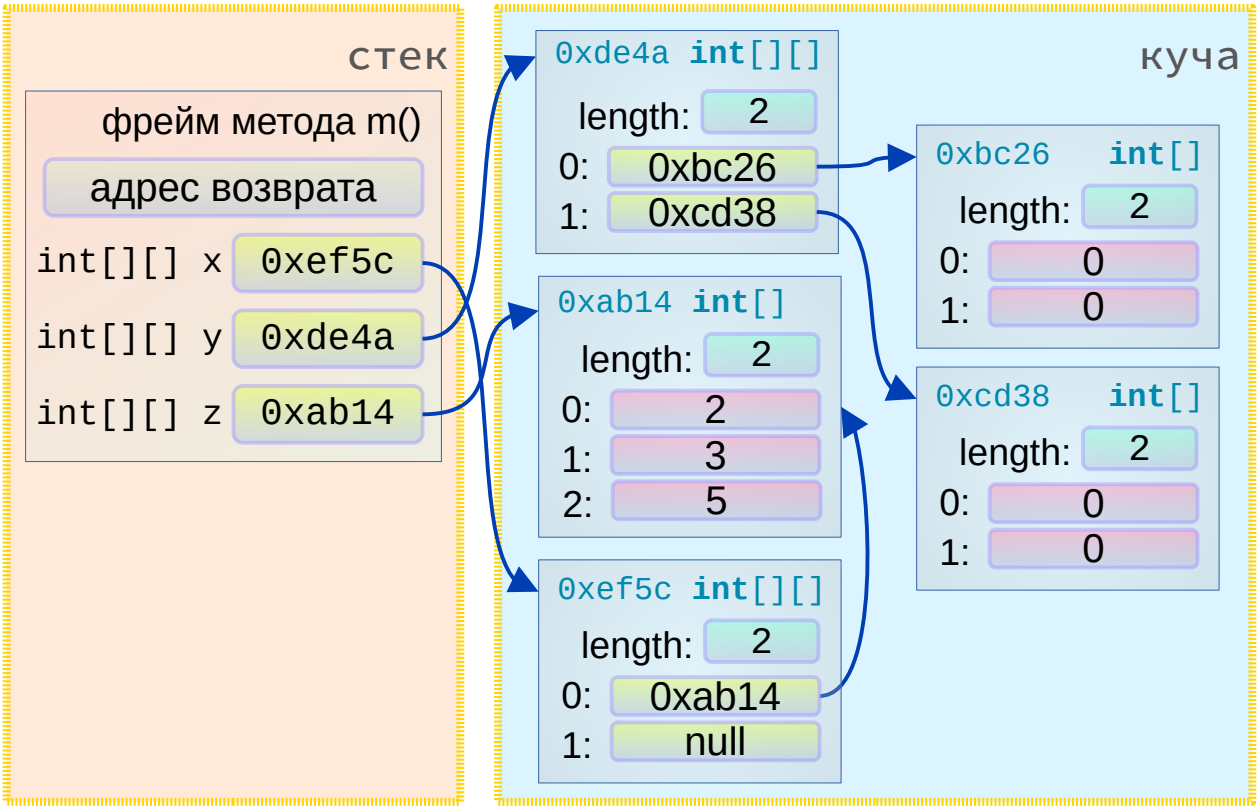
ИТМО

```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



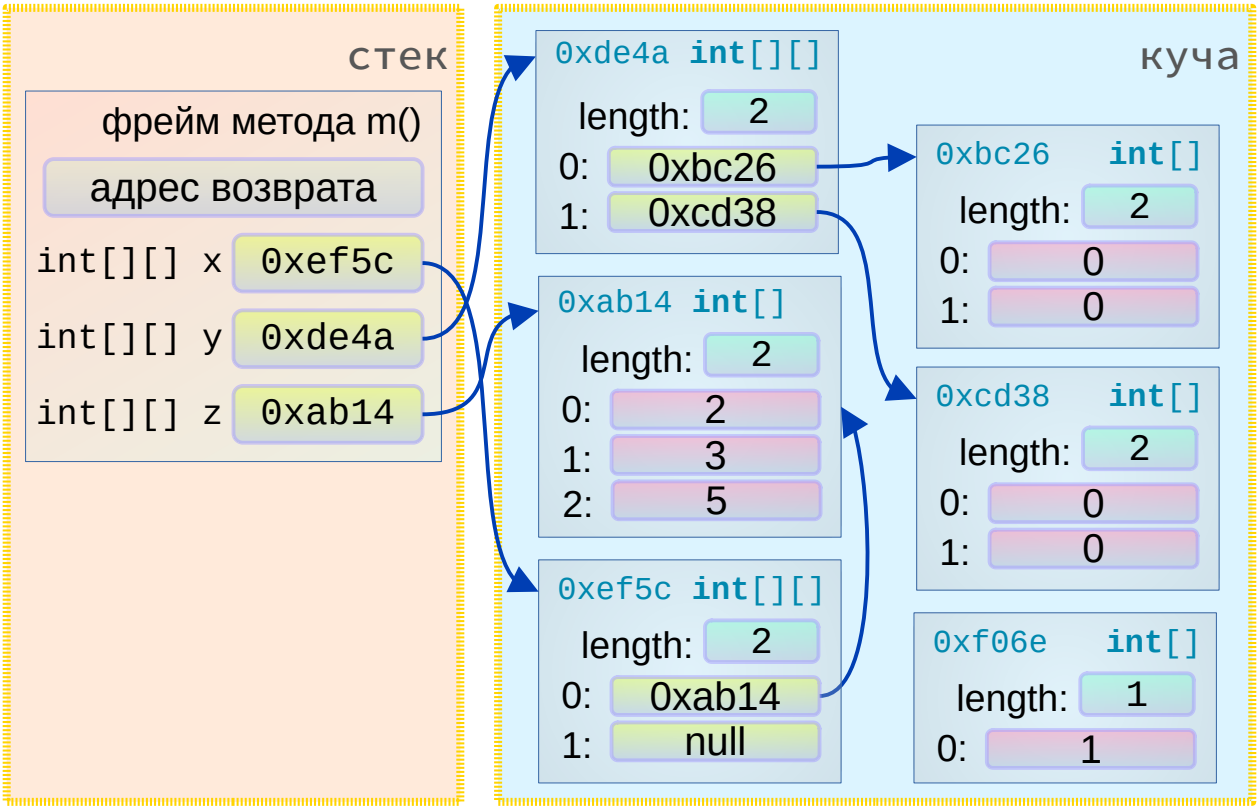
Двумерные массивы в памяти

```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



Двумерные массивы в памяти

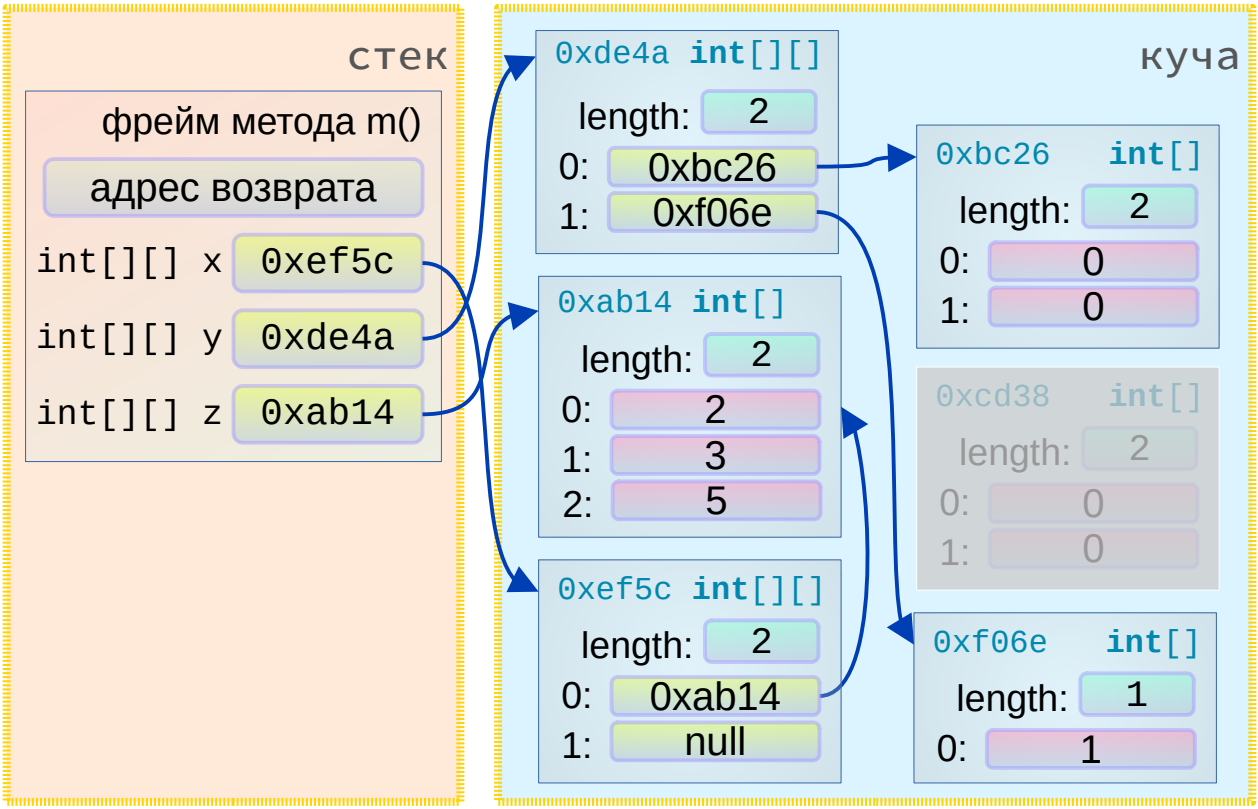
```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



Двумерные массивы в памяти

ИТМО

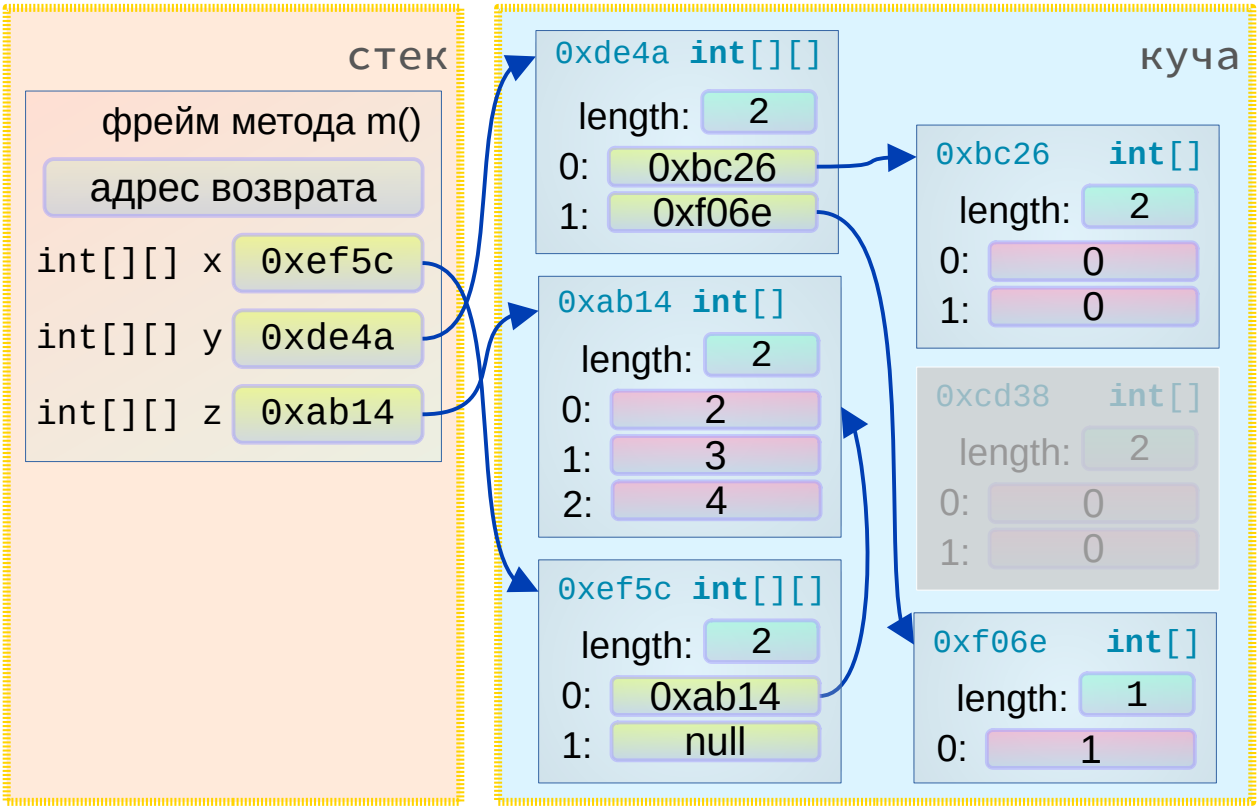
```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



Двумерные массивы в памяти

ИТМО

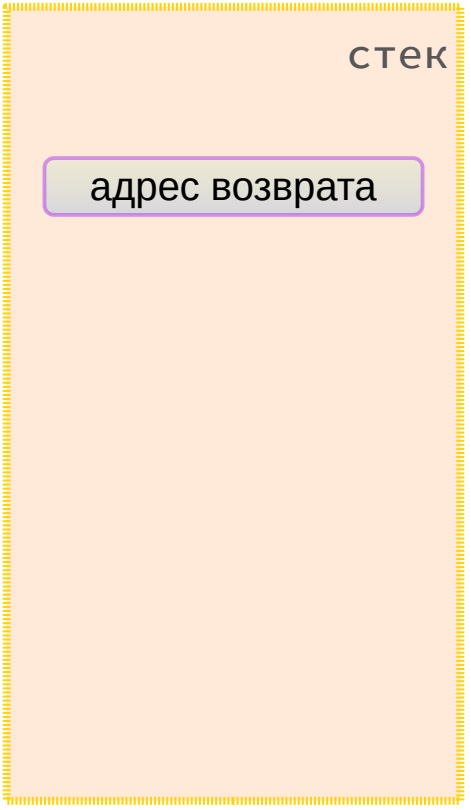
```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```



Двумерные массивы в памяти

ИТМО

```
void m() {
  int[][] x;
  int[][] y;
  int[] z = {2,3,5};
  x = new int[2][2];
  y = x;
  x = new int[2][];
  x[0] = z;
  y[1] = new int[]{1};
  z[2] = 4;
}
```

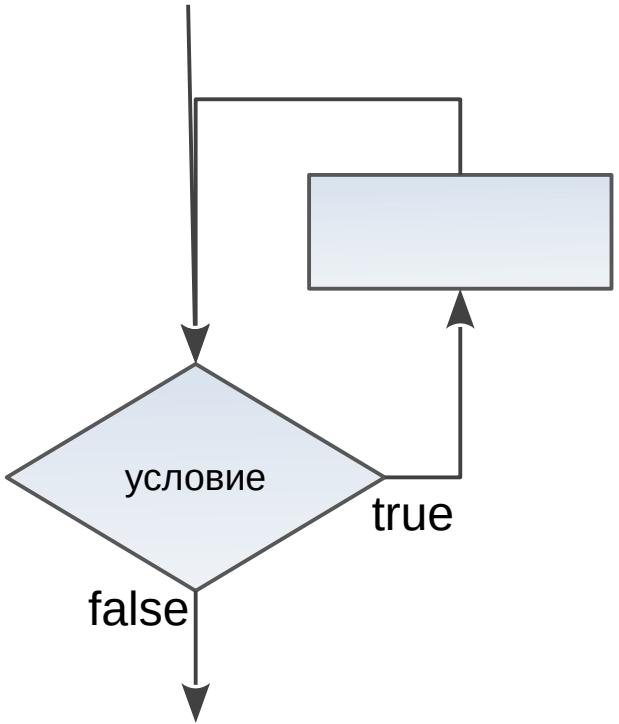


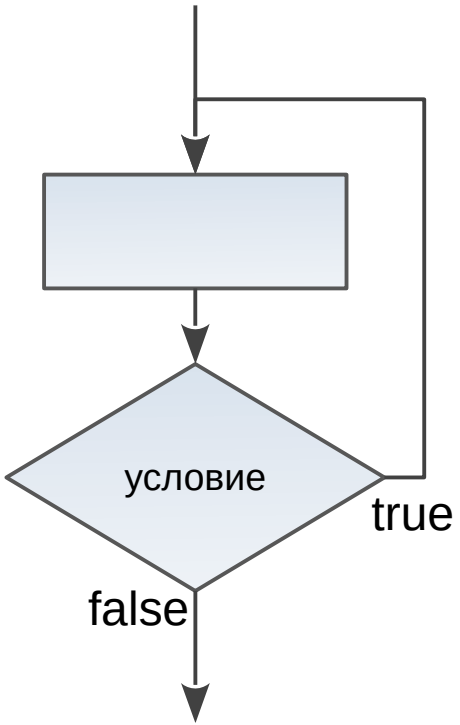
Еще о массивах

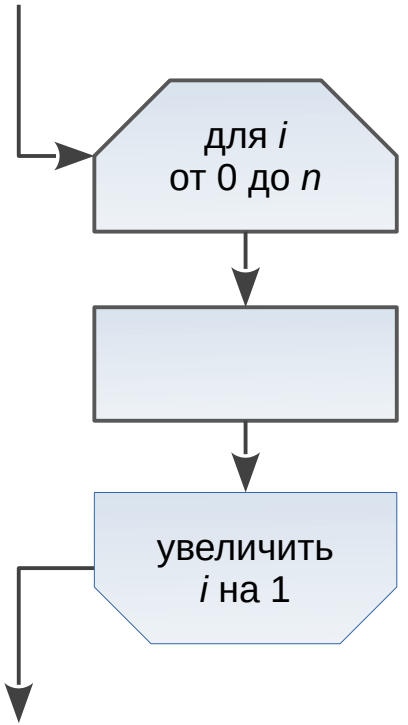
ИТМО

- При сравнении массивов с помощью == сравниваются ссылки на массивы - true, если это один и тот же массив
- Присваивание не создает полную копию массива, копируется только ссылка (shallow copy)
- Для сравнения элементов массивов, для глубокого копирования, сортировки и других действий с массивами можно использовать методы класса java.util.Arrays

Циклы


условие
true
false


условие
true
false


для i
от 0 до n
увеличить
i на 1

Циклы

ИТМО

- Цикл позволяет исключить дублирование кода
- Переменные могут параметризовать цикл
- Цмул:
  - с предусловием,
  - с постусловием,
  - со счетчиком

```
int[] arr = {1,2,3};
int sum = 0;
sum += arr[0];
sum += arr[1];
sum += arr[2];

int sum = 0;
for (int i : arr) {
  sum += i;
}
```

Циклы while и do-while

ИТМО

```
/* while (condition)
      statement;
 Цикл с предусловием
*/
int x = 100;
while (x > 0)
    x -= 1;
```

```
/* do statement;
   while (condition);
 Цикл с постусловием
*/
int x = 100;
do x -= 1;
while (x > 0);
```

Для надежности - блоки

ИТМО

```
/* while (condition)
      statement;
 Цикл с предусловием
*/
int x = 100;
while (x > 0) {
    x -= 1;
}
```

```
/* do statement;
   while (condition);
 Цикл с постусловием
*/
int x = 100;
do {
  x -= 1;
} while (x > 0);
```

while и do-while

- while
  - Если условие сразу ложно, цикл не выполнится ни разу
- do
  - Цикл выполнится хотя бы один раз в любом случае
  - Удобно для валидации ввода пользователя

Цикл for со счетчиком

```
for (int i = 0; i < 10; i++) {
   System.out.print(i);
}
/* for (before;condition;increment)
      statement;
*/
```

- Замена цикла while в подходящих случаях
- Более компактная запись - все в заголовке
- Иногда можно обойтись пустым циклом

```
int i = 0;
while (i < 10) {
   System.out.print(i);
   i++;
}
```

Цикл for для каждого элемента массива

ИТМО

```
int daysInYear = 0;
for (int i=0;i<byMonth.length;i++) {
  daysInYear += byMonth[i];
  System.out.println(byMonth[i]);
}
System.out.print("Total:    ");
System.out.println(daysInYear);
```

```
int[] byMonth =
  {31,28,31,30,31,30,
   31,31,30,31,30,31};
```

Цикл for для каждого элемента массива

ИТМО

```
int daysInYear = 0;
for (int days : byMonth) {
  daysInYear += days;
  System.out.println(days);
}
System.out.print("Total:    ");
System.out.println(daysInYear);
```

```
int[] byMonth =
  {31,28,31,30,31,30,
   31,31,30,31,30,31};


/*
  for (type element : array)
    statement;
*/
```

break и coninue

ИТМО

- break
  - выходит из текущего цикла
  - текущая итерация не завершается
  - может использоваться с меткой для вложенных циклов
- continue
  - переходит к следующей итерации
  - текущая итерация не завершается
  - может использоваться с меткой для вложенных циклов

continue

```
// Сумма ненулевых элементов
int[] array = {9,0,3,6,0,...};
int sum = 0;
for (int e : array) {
  if (e != 0) {
    sum += e;
  }
}
System.out.println(sum);
```

continue :)

ИТМО

```
// Сумма ненулевых элементов
int[] array = {9,0,3,6,0,...};
int sum = 0;
for (int e : array) {
  if (e != 0) {
    sum += e;
  }
}
System.out.println(sum);
```

```
// Сумма ненулевых элементов
int[] array = {9,0,3,6,0,...};
int sum = 0;
for (int e : array) {
  sum += e;
}
System.out.println(sum);
```

continue

```
// Сумма элементов, которые не
делятся на 3 и на 5
int[] array = {8,0,3,5,7,...};
int sum = 0;
for (int e : array) {
  if (e % 3 != 0) {
    if (e % 5 != 0) {
      sum += e;
    }
  }
}
System.out.println(sum);
```

```
// Сумма элементов, которые не
делятся на 3 и на 5
int[] array = {8,0,3,5,7,...};
int sum = 0;
for (int e : array) {
  if (e % 3 == 0) continue;
  if (e % 5 == 0) continue;
  sum += e;
}
System.out.println(sum);
```

break

```
// Баллы за семестр
int grades = {...};
double sum = 0.0;
for (int i=0;
     i<grades.length && sum<60;
     i++) {
  sum += grades[i];
}
System.out.println(sum < 60 ?
  "Failed" : "Passed");
```

```
// Баллы за семестр
int grades = {...};
double sum = 0.0;
for (double grade : grades) {
  sum += grade;
  if (sum >= 60) break;
}
System.out.println(sum < 60 ?
  "Failed" : "Passed");
```

Вложенные циклы

ИТМО

```
// Bubble sort
int[] array = { ... };
for (int i=1; i<array.length; i++) {
  for (int j=0; j<i; j++) {
    if (array[i] < array[j]) {
      int tmp = array[i];
      array[i] = array[j];
      array[j] = tmp;
    }
  }
}
```

```
// Print matrix
int[][] array = {{...},{...}};
for (int[] inner : array) {
  for (int element : inner) {
    System.out.print(element+" ");
  }
  System.out.println();
}
```

break и continue с метками

ИТМО

```
outer:
for (int[] inner : array) {
  for (int element : array) {
    if (...) continue;
    if (...) continue outer;
    if (...) break outer;
    if (...) break;
  }
}
```

String

- String содержит массив символов
  - типа byte (если символы строки входят в LATIN-1) - compact String
  - типа char (если есть символы, не входящие в LATIN-1)
- Строки - неизменяемые (любое измеение - это новая строка)
- Текстовые литералы хранятся в куче в пуле строк
- Если такая строка уже есть в пуле, новая не создается - интернирование строк.
- Сравнение строк: == (одна и та же); equals (то же содержимое)

Строки

ИТМО

```
String s1 = "Hello world!";
// s1 == "Hello world!";
String s2 = "Hello" + " world!";
// s1 == s2
String s3 =
  new String("Hello world!");
// s1 != s3
s3 = s3.intern();
// s1 == s3
```

```
String s4 = "Hello";
String s5 = "H"+"e"+"l"+"l"+"o";
// s4 == s5
s4 += " world!";
// s4 != s5
// s4 != s1
// s4.equals(s1)
s4 = s4.intern();
// s4 == s1
```

String

- Корректное создание строк - текстовый литерал
- Не нужно создавать строки с помощью new
- Не рекомендуется в цикле склеивать строки - создается много новых объектов
- Для изменяемых строк есть класс StringBuilder с поддержкой вставки, добавления, удаления символов

Методы (пока статические)

ИТМО

- Процедурная парадигма
  - Программа состоит из процедур и функций, каждая из которых выполняет свою задачу
  - Исключение повторения кода - меньше писать, проще поддерживать

```
System.out.println("Hello world");
System.out.println("Hello cat");
...
System.out.prinltn("Hello dog");

void hello(String name) {
  System.out.println("Hello " +name);
}
hello("world");
hello("cat");
...
hello("dog");
```

Методы (подпрограммы, функции, ...)

- Подпрограмма - отдельно вызываемая часть программы для выполнения какой-то подзадачи
- Процедура - подпрограмма, выполняющая действия
- Функция - подпрограмма, вычисляющая значение
- Метод - подпрограмма, связанная с классом или объектом.

Методы (подпрограммы, функции, ...)

ИТМО

```
public class Hello {
  public static void hello(String name) {
    System.out.println("Hello "+name+"!");
  }
}
/* modifiers return_type name([ par_type par_name ]) {
     statements;
   }
```

Методы (подпрограммы, функции, ...)

ИТМО

```
public class Hello {
  public static void hello(String name) {
    System.out.println("Hello "+name+"!");
  }
}
```

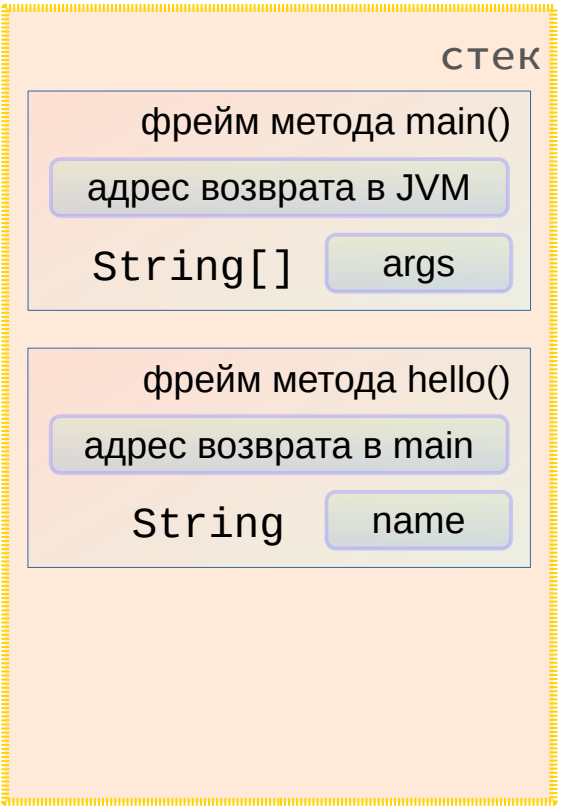
сигнатура

```
/* modifiers return_type name([ par_type par_name ]) {
     statements;
   }
```

Методы (подпрограммы, функции, ...)

ИТМО

```
public class Hello {
  public static void hello(String name) {
    System.out.println("Hello "+name+"!");
  }
  public static void main(String[] args) {
    hello("world");
// внутри метода hello() name = "world";
  }
}
```



Методы в Java

ИТМО

- Аргументы передаются в метод по значению
- Вызывающий метод кладет в стек копии значений аргументов
- Внутри метода эти значения доступны по именам параметров
- Если параметр примитивного типа, передается копия значения
- Если параметр ссылочного типа, передается копия ссылки. Объект при этом - в куче. Ссылку изменить нельзя, но объект иногда можно

Методы в Java

ИТМО

- Если метод не должен возвращать значение, то тип возвращаемого значения - void
- Если метод должен возвращать значение определенного типа, то последней инструкцией метода должен быть возврат значения заданного типа с помощью return
- Для параметров и возвращаемых значений действуют правила приведения типов

Параметры и возвращаемое значение

ИТМО

```
public static String plural(int num,
                            String word,
                            String decl) {
  int index = num/10%10 == 1 || (num+9)%10 > 3 ?
          2 : num%10 > 1 ? 1 : 0;
  String[] endings = decl.split("|");
  String result = num + " " + word + endings[index];
  return result;
}

System.out.print(plural(24, "студент", "|а|ов") // 24 студента
```

Локальные переменные и var

ИТМО

```
public static String plural(int num,
                            String word,
                            String decl) {
  var index = num/10%10 == 1 || (num+9)%10 > 3 ?
          2 : num%10 > 1 ? 1 : 0;
  var endings = decl.split("|");
  var result = num + " " + word + endings[index];
  return result;
}

System.out.print(plural(24, "студент", "|а|ов") // 24 студента
```

Локальные переменные и var

- Локальные переменные - объявленные внутри блоки или метода
- var можно использовать для локальных переменных, которые сразу инициализируются
- var нельзя использовать для массивов
- var нельзя использовать для параметров метода и задания типа возвращаемого значения в заголовке метода
- Тип переменной выводится компилятором из ее значения

Параметр метода main(String[] args)

ИТМО

```
java Hello Петя Вася "Иван Иванович"
           args[0] args[1]      args[2]

String[] args = {"Петя", "Вася", "Иван Иванович"};


public static void main(String[] args) {
  for (String name : args) {
    System.out.println("Hello " + name + "!");
  }
}
```

Переменное число параметров - varargs

ИТМО

```
void main(String[] args) {
  // args – массив строк
}
String[] x = {"A","B","C"}
// Вызов метода
main(x);
main(new String[2]);
```

```
void main(String... args) {
  // args – массив строк
}
String[] x = {"A","B","C"}
// Вызов метода
main(x);
main(new String[2]);
```

Переменное число параметров - varargs

ИТМО

```
void main(String[] args) {
  // args - массив строк
}
String[] x = {"A","B","C"}
// Вызов метода
main(x);
main(new String[2]);


_
```

```
void main(String... args) {
  // args - массив строк
}
String[] x = {"A","B","C"}
// Вызов метода
main(x);
main(new String[2]);
main()
main("world")
main("Pete","John","Bob");
```

Переменное число параметров - varargs

ИТМО

```
void main(String[] args) {
  // args - массив строк
}
String[] x = {"A","B","C"}
// Вызов метода
main(x);
main(new String[2]);
~~main()~~
~~main("world")~~
~~main("Pete","John","Bob");~~
```

```
void main(String... args) {
  // args - массив строк
}
String[] x = {"A","B","C"}
// Вызов метода
main(x);
main(new String[2]);
main()
main("world")
main("Pete","John","Bob");
```

Переменное число параметров - varargs

```
void m(int a, int... x) {
// int... x – в конце
}

m();
m(5);
m(1, 3);
m(0, 2, 4);
m(3, new int[0]);
```

```
void main(String... args) {
  // args – массив строк
}
String[] x = {"A","B","C"}
// Вызов метода
main(x);
main(new String[2]);
main()
main("world")
main("Pete","John","Bob");
```

Перегрузка методов

- Методы с одинаковым именем, но с разной сигнатурой - перегруженные методы (overloaded)
- Если различие только в типе возвращаемого значения - это не перегрузка (так нельзя)
- Компилятор должен быть способен по типу параметров выбрать подходящий метод на этапе компиляции

Перегрузка методов

ИТМО

```
double[] x = { Math.PI };
String hello = "Hello";
System.out.print(x.length);
System.out.print(x[0]);
System.out.print(x[0] < 3);
System.out.print(x);
System.out.print(hello);
System.out.print(hello.charAt(2));
```

Вопросы?