

*Лабораторная №1*  
***Алгоритмы линейного  
поиска***

# Элементы синтаксиса языка Java

# Hello World!

```
4 ▶ public class train {  
5 ▶     public static void main(String[] args) {  
6     |         System.out.println("Hello World!");  
7     |     }  
8     }  
    }
```

- Любая программа обязательно содержит класс.
- Public указывает на доступность класса другим частям программы.
- Фигурные скобки называются операторами блока.
- Метод main – точка входа в программу.
- Статический метод можно использовать без объектов.
- Void – не имеет возвращаемого значения.

# Hello World!

```
4 ▶ public class train {  
5 ▶     public static void main(String[] args) {  
6     |         System.out.println("Hello World!");  
7     |     }  
8     }  
    }
```

- `String[] args` – аргумент командной строки (массив строковых данных).
- `System.out.println` – ГОТОВЫЙ МЕТОД, осуществляющий вывод надписи на экран.

# Примитивные типы данных

Тип данных	Размер, байт	Допустимые значения
byte	1	$-128 \div 127$
short	2	$-32768 \div 32767$
int	4	$-2147483648 \div 2147483647$
long	8	$-2^{63} \div 2^{63} - 1$
float	8	$\approx 1.4 \cdot 10^{-45} \div 3.4 \cdot 10^{38}$
double	16	$\approx 1.7 \cdot 10^{-324} \div 4.9 \cdot 10^{308}$
char	2	$0 \div 65535$
boolean	—	true, false

Переменные этих типов не являются объектами

# Литералы (константы)

- Целочисленные литералы в десятичном виде записываются непосредственно, но литералы типа `long` должны иметь суффикс «L» или «l» (например, `47L`).
- Литералы типа `float` должны иметь суффикс «F» или «f» (например, `3.5F`), а литералы типа `double` — суффикс «D» или «d», который может опускаться (`3.5D` или просто `3.5`).
- Символьные литералы ограничиваются одинарными кавычками (например, `'ю'`).
- Строковые литералы ограничиваются двойными кавычками (`"abcd"`).
- Два логических литерала `true` и `false`, не равные

# Объявление переменных

Любая переменная может быть инициализирована в момент описания любым допустимым выражением.

Например:

- `int i = 10;`
- `float f = 3.5f;`
- `char x = 'f';`
- `boolean b = false;`

Заметьте, что строка должна заканчиваться точкой с запятой.

# Массивы

- Массивы в Java являются объектами.
- Объявление одномерного массива: к имени переменной либо к имени типа данных добавляются пустые квадратные скобки, например `int a[];` или `int[] b;`
- Для создания самого массива (объекта класса «массив», а значит и выделения на него памяти) следует использовать операцию `new`: `a = new int[50];`
- Возможно одновременное объявление и выделение памяти: `int a[] = new int[50];`
- Возможна также инициализация массива при его создании: `int a[] = { 20, 50, 166, 72, 0, -53 };`



# Работа с массивами

- Для обращения к элементам массива используется операция [].
- Элементы массива нумеруются с нуля.
- Размеры массива определяются посредством переменной `length`, вызываемой как метод объекта класса «массив», например `a.length` возвращает длину массива `a`.
- Многомерные массивы представляют собой массивы массивов и создаются аналогично одномерным, например: `int a[][] = new int[5][2];`

# Арифметические операции

- унарные операции сохранения (+) и изменения знака (-);
- унарные операции инкремента (++) и декремента (--);
- бинарные операции сложения (+), вычитания (-), умножения (\*), деления (/) и нахождения остатка от деления (%);
- операции с присваиванием (+=, -=, \*=, /=, %=).  
Например, `a+=16` эквивалентно `a=a+16`.
- Арифметические операции применяются к числовым типам и имеют результат также числового типа.
- Операция деления для целочисленных типов даёт результат целочисленного типа (неполное частное)

# Операции сравнения

- Операции `==`, `!=`, `<`, `>`, `<=`, `>=`.
- Их результат всегда имеет тип `boolean`.

# Логические операции

- унарная операция логического «не» (`!`),
- бинарные операции логического «и» (`&&`, `&`), «или» (`|`, `||`) и «исключающего или» (`^`),
- аналогичные операции с присваиванием (`&=`, `|=`, `^=`).
- Логические операции применяются к аргументам типа `boolean` и дают результат типа `boolean`.

# Условный оператор if

```
if (<условие>
{
<операторы>}
else {
<операторы>}
```

Условие должно иметь  
тип boolean

```
4 public class train {
5     public static void main(String[] args) {
6         int i = 10;
7         if(i*i > 50){
8             System.out.println("Тест прошёл не очень успешно.");}
9         else{
10            System.out.println("Тест прошёл успешно.");}
11     }
12 }
```

# Операторы цикла

```
for(int i = 0; i < N; ++i) {  
<операторы>}
```

Объявленные в операторе переменные действительны лишь внутри цикла

```
while  
(<условие>){  
<операторы>}
```

```
do{  
<операторы>  
} while  
(<условие>)
```

оператор постусловия (сначала выполнит, потом проверит), т.е. даже если условие не выполняется никогда, один раз действие выполнено будет

# Комментарии

- можно создавать с помощью двух слешей в конце строки (//)
- или с помощью конструкции «/\* .... \*/».

# Алгоритмы линейного поиска

# Линейный поиск

- *Дано*: массив  $A$  с  $n$  элементами.
- *Выяснить*: присутствует ли в массиве  $A$  значение  $x$ . Если да, то мы хотим знать индекс  $i$ , такой, что  $A[i] = x$ . Если нет – вывести соответствующее сообщение. При этом  $x$  может встретиться в массиве более одного раза.
- *Алгоритм линейного поиска*: мы начинаем с начала массива (первого его элемента), поочередно проверяя все его элементы ( $A[0]$  затем  $A[1]$ , затем  $A[2]$  и так далее до  $A[n-1]$ ) и записывая место, где мы находим  $x$  (если мы вообще находим его).



# Процедура линейного поиска

## Процедура `Linear-Search(A,n,x)`

*Вход:*

- $A$  – массив;
- $n$  – количество элементов массива  $A$ , среди которых выполняется поиск;
- $x$  – искомое значение.

*Выход:* значение переменной `answer` – либо индекс  $i$ , для которого  $A[i] = x$ , либо специальное значение `not-found`, которое может представлять собой любой некорректный индекс массива, например, произвольное отрицательное значение.





*Шаги процедуры:*

1. Установить значение `answer` равным `not-found`.
2. Для каждого индекса  $i$ , пробегающего поочередно значения от 0 до  $n-1$ :
  - A. Если  $A[i] = x$ , установить значение `answer` равным  $i$ .
3. В качестве выходного вернуть значение `answer`.

# Улучшенный линейный поиск

Прекращаем поиск, как только он находит в массиве значение  $x$ .

## **Процедура Better-Linear-Search( $A, n, x$ )**

*Вход и выход:* те же, что и в Linear-Search.

*Шаги процедуры:*

1. Для  $i = 0$  до  $n-1$ :

    А. Если  $A[i] = x$ , вернуть значение  $i$  в качестве выхода процедуры.

2. Вернуть в качестве выходного значение not-found.

# Поиск с ограничителем

Цель – избежать двойной проверки при каждой итерации цикла. Для этого поместим искомое значение  $x$  в последнюю позицию  $A[n]$  после сохранения содержимого  $A[n]$  в другой переменной. Когда мы находим  $x$ , мы выполняем проверку, действительно ли мы его нашли или просто достигли конца массива. Значение, которое мы помещаем в массив, называется *ограничителем*.

## Процедура Sentinel-Linear-Search( $A, n, x$ )

*Вход и выход:* те же, что и в Linear-Search.

*Шаги процедуры:*

1. Сохранить  $A[n-1]$  в переменной last и поместить  $x$  в  $A[n-1]$ .
2. Установить  $i$  равным 0.
3. Пока  $A[i] \neq x$ , увеличивать  $i$  на 1.
4. Восстановить  $A[n-1]$  из переменной last.
5. Если  $i < n-1$  или  $A[n-1] = x$ , вернуть значение  $i$  в качестве выхода процедуры.
6. В противном случае вернуть в качестве возвращаемого значения not-found.

# Задание

1) Освоиться с синтаксисом Java.

2) Реализовать три рассмотренных алгоритма линейного поиска на языке Java.