

Основы языка программирования Java

Компилируемые языки программирования

- **Компиляция** – преобразование текста программы, написанного на языке высокого уровня (C, C++, Pascal), в набор инструкций, которые может выполнять исполнитель
- Скомпилированная программа часто называется **байт-кодом**

Компилируемые языки программирования

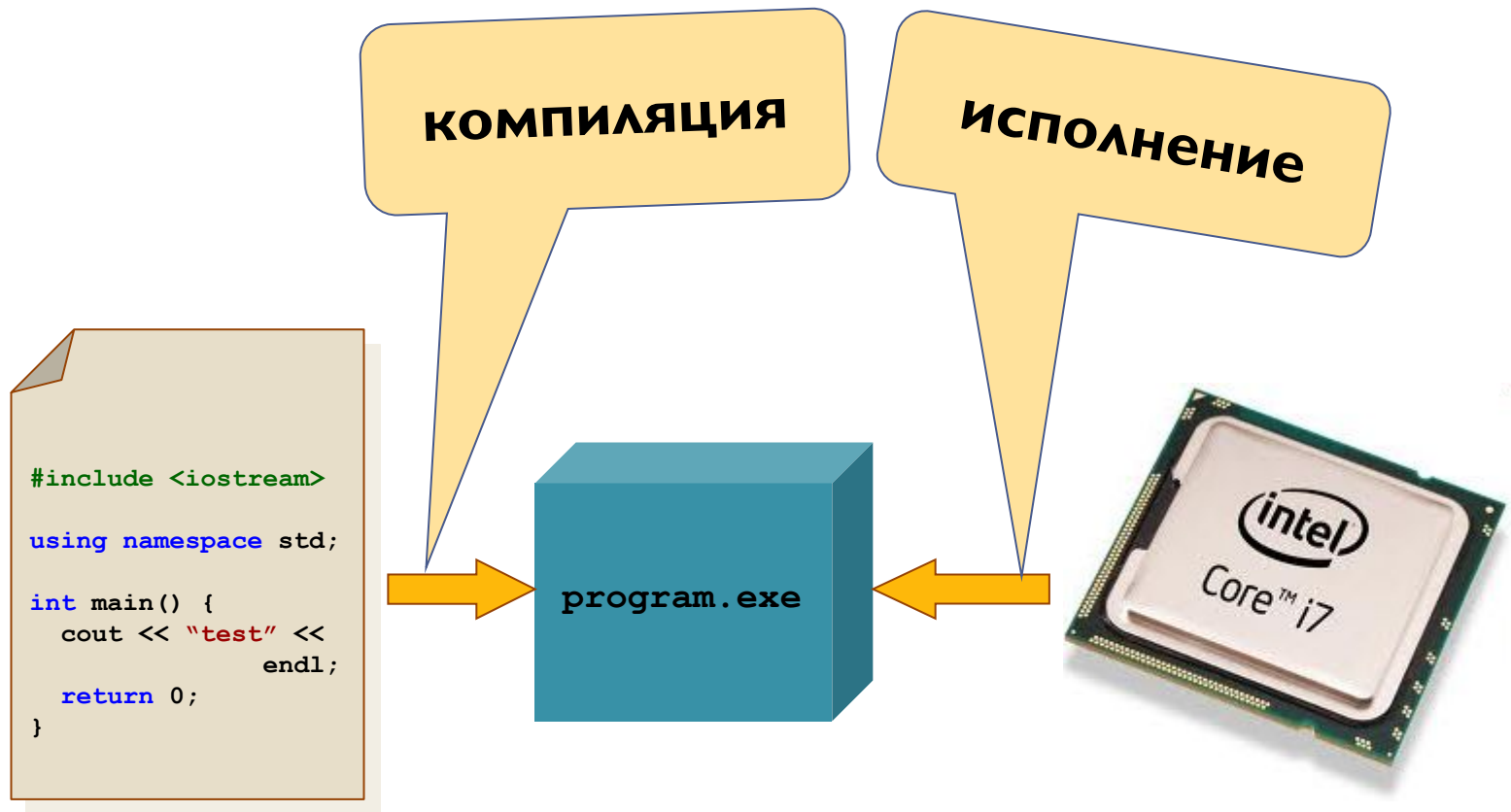
Плюсы:

- Высокая скорость исполнения программы
- Отсутствие необходимости в дополнительном программном обеспечении для запуска программы

Минусы:

- Привязка к исполнителю

Компилируемые языки программирования



Интерпретируемые языки программирования

- **Интерпретация** – анализ текста программы, написанного на языке высокого уровня (JavaScript, PHP), и непосредственное исполнение обнаруженных инструкций
- Интерпретатор является исполнителем

Интерпретируемые языки программирования

Плюсы:

- Кроссплатформенность

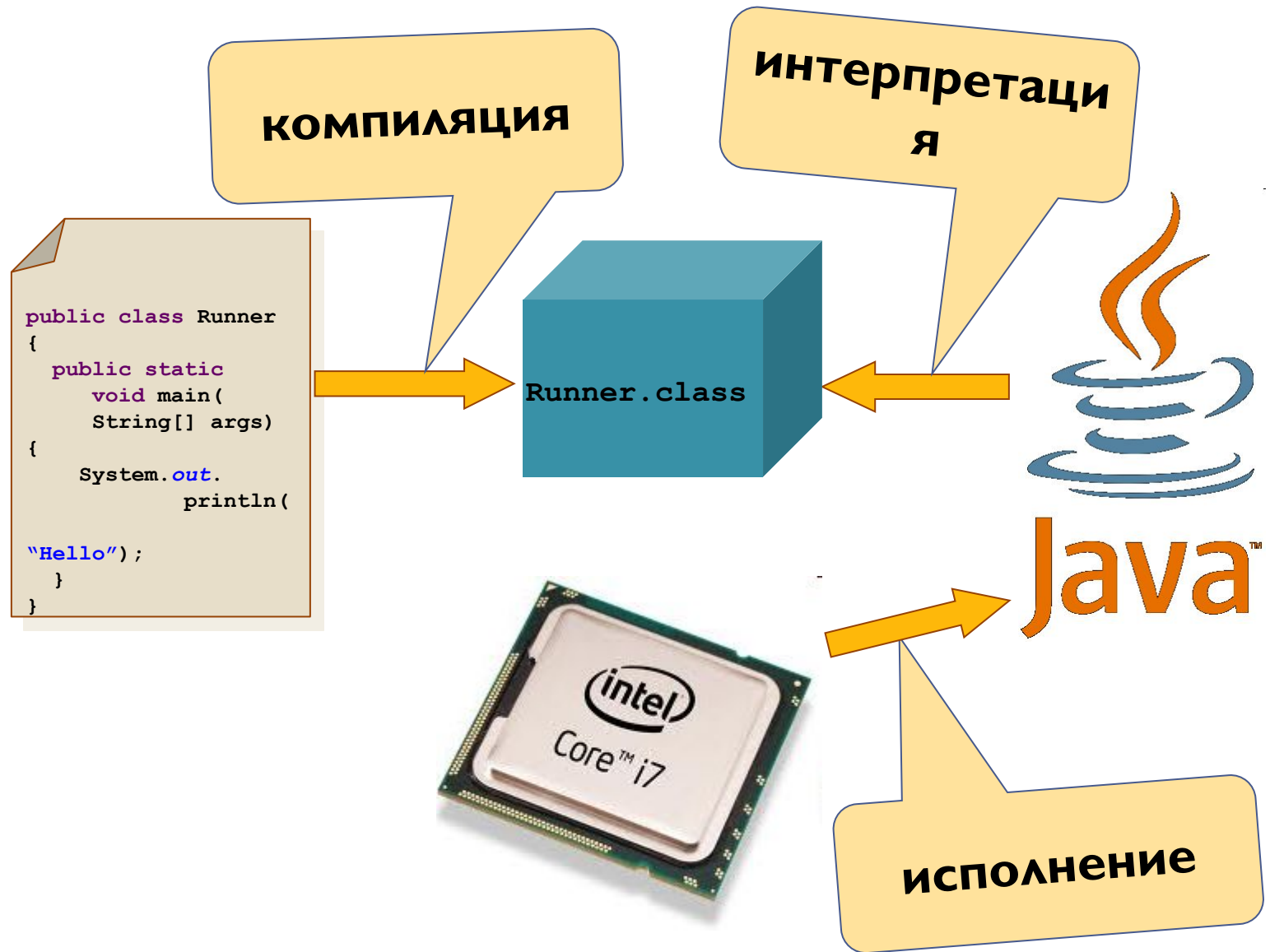
Минусы:

- Для запуска необходима дополнительная программа-интерпретатор
- Низкая скорость работы

Интерпретируемые языки программирования



Язык программирования Java



Область применения

- Консольные приложения
- Настольные приложения (AWT, Swing)
- Апплеты
- Мидлеты
- Сервлеты

Отличия от C++

- Отсутствие указателей и прямого доступа к памяти
- Механизм сборки мусора
- Отсутствуют деструкторы
- Неотделимость спецификации класса от реализации
- Отсутствие перегрузки операторов

Отличия от C++

- Отсутствие структур и объединений
- Отсутствуют шаблоны (templates), вместо них используют обобщения (generics)
- Отсутствуют беззнаковые целые числа
- Отсутствуют аргументы по умолчанию
- Не используются `goto` и `const`

Первая программа

```
// D:\java\proj\by\vsu\HelloWorld.java
```

```
package by.vsu;
```

```
public class HelloWorld {
```

```
    public static void main(String[]
```

```
        args) {
```

```
        System.out.println("Hello World");
```

```
    }
```

```
}
```

Компиляция программы

```
C:\Users\user> _
```

Компиляция программы

```
C:\Users\user> D:
```

```
D:> _
```

Компиляция программы

```
C:\Users\user> D:
```

```
D:> cd java\proj
```

```
D:\java\proj> _
```

Компиляция программы

```
C:\Users\user> D:
```

```
D:> cd java\proj
```

```
D:\java\proj> javac by\vsu\HelloWorld.java
```

```
D:\java\proj> _
```


Запуск программы

```
C:\Users\user> D:
```

```
D:> cd java\proj
```

```
D:\java\proj> javac by\vsu\HelloWorld.java
```

```
D:\java\proj> java by.vsu.HelloWorld
```

```
Hello World
```

```
D:\java\proj> _
```

Ввод данных

```
// ReadChar.java
```

```
public class ReadChar {  
    public static void main(String[]  
        args) {  
        int x;  
        try {  
            x = System.in.read();  
            char c = (char) x;  
        } catch(java.io.IOException e) {}  
    }  
}
```

Типы данных

Тип	Размер (бит)	Значения
boolean	8	true, false
byte	8	-128..127
short	16	-32 768..32 767
int	32	-2 147 483 648..2 147 483 647
long	64	-9 223 372 036 854 775 808.. 9 223 372 036 854 775 807
char	16	'\u0000'..' \uffff'
float	32	3.40282347E+38
double	64	1.797693134486231570E+308

Целочисленные литералы

- 23 – десятичное число
- 012 – восьмеричное число
- 0x7a – шестнадцатеричное число

тип литералов по умолчанию **int**

литерал типа **long** обозначается **L**

- 78L

Дробные литералы

- 1.234
- 0.123E-03

литералы типа **double**

литерал типа **float** обозначается **F**

- 2.34F

Символьные литералы

- `'a'`
- `'\123'`
- `'\u9ae8'`
- `'\n'`
- `'\t'`

Идентификаторы

- Не могут начинаться с цифры
- Не могут содержать знаки арифметических и логических операторов
- Не могут содержать символ '#'

Переменные

- **int a;**
- **char b = '#';**

Область видимости

```
// операторы (1) – x недоступна
while(a < 10) {
    // операторы (2) – x недоступна
    int x;
    // операторы (3)
    if(b > 0) {
        // операторы (4)
    }
    // операторы (5)
}
// операторы (6) – x недоступна
```

Арифметические операторы

+	Сложение	+ =	Сложение с присваиванием
-	Вычитание	- =	Вычитание с присваиванием
*	Умножение	* =	Умножение с присваиванием
/	Деление	/ =	Деление с присваиванием
%	Остаток от деления	% =	Остаток от деления с присваиванием
++	Инкремент	++	Декремент

Операторы сравнения

<	Меньше	>	Больше
<=	Меньше или равно	>=	Больше или равно
==	Равно	!=	Не равно

Логические операторы

&	И
&&	сокращённое И
 	ИЛИ
 	сокращённое ИЛИ
!	НЕ

Пример логических операторов

```
int a = 1, b = 0;
```

```
int c = 1, d = 1;
```

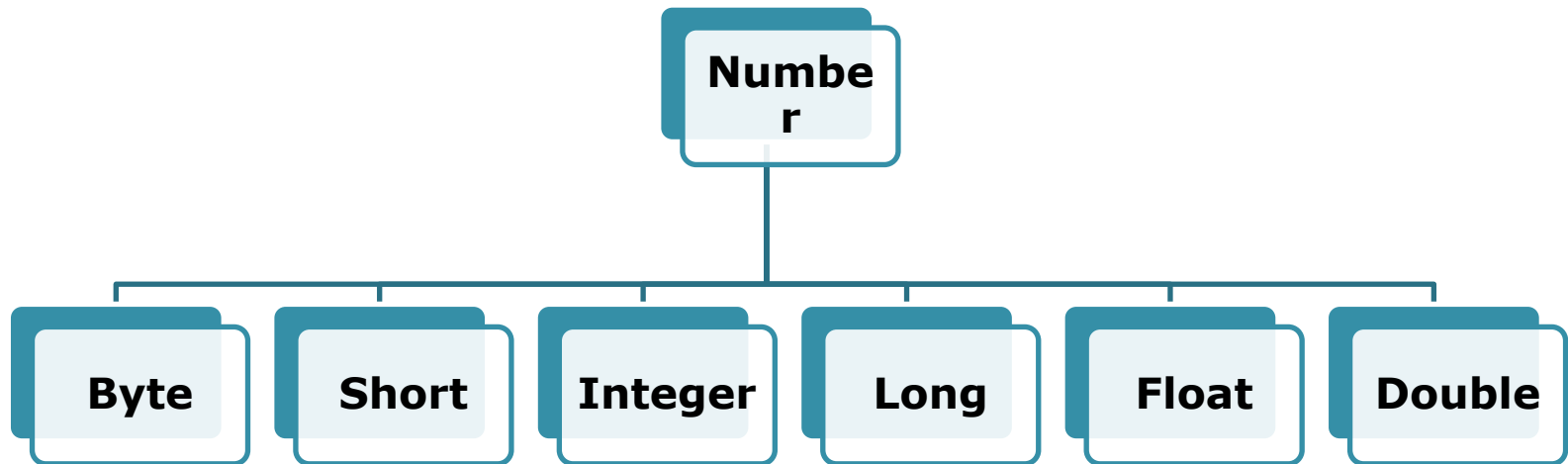
```
a == b & c == d++ // d = 2
```

```
a == b && c == d++ // d =  
1
```

Классы-оболочки

Тип	Класс
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double

Иерархия классов-оболочек



Математические константы

- **Math.** ***π***
- **Math.** ***e***

Математические методы

- **Math.*abs*(x);**
- **Math.*sqrt*(x);**
- **Math.*cbrt*(x);**
- **Math.*pow*(x, y);**
- **Math.*hypot*(x, y);**

Математические методы

- **Math.*cos*(x);**
- **Math.*sin*(x);**
- **Math.*tan*(x);**

Математические методы

- **Math.acos(x);**
- **Math.asin(x);**
- **Math.atan(x);**
- **Math.atan2(y, x);**

Математические методы

- ***Math.toDegrees(x);***
- ***Math.toRadians(x);***

Математические методы

- **Math.*cosh*(x);**
- **Math.*sinh*(x);**
- **Math.*tanh*(x);**

Математические методы

- ***Math.exp(x);***
- ***Math.log(x);***
- ***Math.log10(x);***

Математические методы

- **Math.*ceil*(x);**
- **Math.*floor*(x);**
- **Math.*round*(x);**

Математические методы

- **`Math.random();`**
- **`Math.min(x, y);`**
- **`Math.max(x, y);`**

Массивы. Объявление массива

<тип> <имя переменной> [];

Или

<тип> [] <имя переменной>;

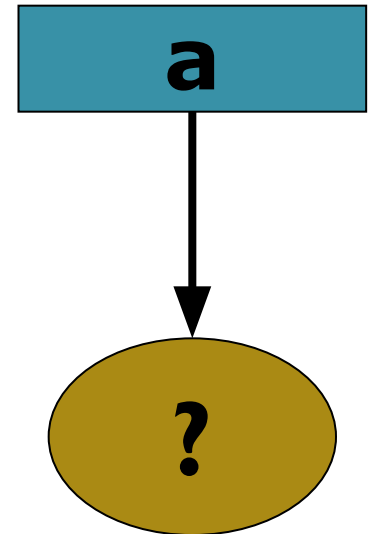
Пример объявления массивов

```
int a[];
```

```
double[] b;
```

```
int a[], b, c[], d;
```

```
int[] a, b, c[], d;
```



Массивы. Создание массива

```
new <тип> [<размер>];
```

Пример создания массива

```
int a[];
```

```
...
```

```
a = new int[5];
```

```
int a[] = new int[5];
```



Обращение к элементу массива

```
System.out.println(a[6]);
```

```
a[4] = 234;
```

Заполнение массива

```
double array[] = new double[10];
```

```
for(int i = 0; i < 10; i++) {  
    array[i] = 10 * Math.random();  
}
```

Вывод массива

```
for(int i = 0; i < 10; i++) {  
    System.out.print(array[i]);  
    System.out.print(' ');  
}
```

Инициализация массива

```
new <тип> [ ] { <список  
значений> }
```


Пример инициализации массива

```
int a[];
```

```
a = new int[] {1, 2, 3};
```

```
char b[] = new char[] {'1', '2'};
```

```
function(new double[] {1.2, 3.4});
```

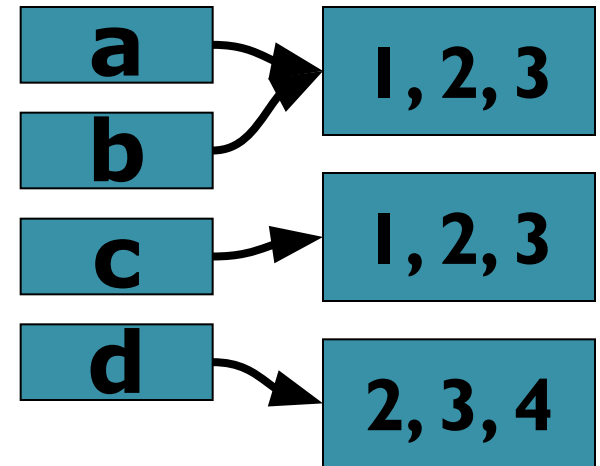
Операции с массивами

```
int a[] = {1, 2, 3};
```

```
int b[] = a;
```

```
int c[] = {1, 2, 3};
```

```
int d[] = {2, 3, 4};
```

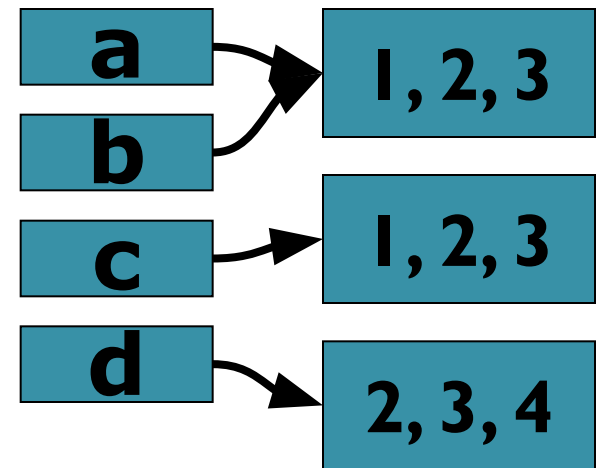


Операции с массивами

a == b // true

a == c // false

a == d // false



Массив как класс

```
int a[] = new int[N];
```

```
...
```

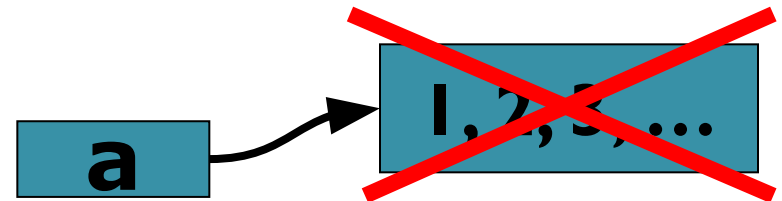
```
int b[] = a.clone(); // a == b  
                // false
```

```
...
```

```
int M = a.length;
```

```
...
```

```
a = null;
```



Сравнение массивов

```
int a[] = { /* элементы */ };
int b[] = { /* элементы */ };
boolean isEqual =
    a.length == b.length;
if(isEqual) {
    for(int i = 0; i < a.length; i++) {
        if(a[i] != b[i]) {
            isEqual = false;
            break;
        }
    }
}
System.out.println(isEqual);
```

Массив массивов

```
int[] array[];
```

тип имя переменная массив

```
int[][] array;
```

или

```
int array[][];
```

Создание двумерных массивов

```
int[][] matrix = new int[56][];
```

```
for(int i = 0; i < matrix.length;  
      i++) {  
    matrix[i] = new int[i+1];  
}
```

Создание и обработка двумерных МАССИВОВ

```
int[][] a = new int[N][M];
```

```
for(int i = 0; i < a.length; i++) {  
    for(int j = 0;  
        j < a[i].length;  
        j++) {  
        System.out.print(a[i][j]);  
        System.out.print('\t');  
    }  
    System.out.println();  
}
```


Инициализация двумерных массивов

```
int[][] a = {  
    {1, 2, 3, 4},  
    {5, 6, 7, 8},  
    {9, 10, 11, 12}  
};
```

Инициализация двумерных массивов

```
int[][] a = {  
    {1},  
    {2, 3},  
    {4, 5, 6},  
    {7, 8, 9, 10},  
    {11, 12},  
    {}  
};
```

Класс

- Имя
- Атрибуты, **Поля**, Переменные
- Операции, **Методы**, Функции

Принципы ООП

- Инкапсуляция
- Наследование
 - Абстракция
- Полиморфизм
 - Раннее связывание
 - Позднее связывание

Описание класса

```
class ArrayAlgorithms {  
    int[] array;  
    void sort(boolean reverse) {  
        // реализация  
    }  
    int max() {  
        // реализация  
        return result;  
    }  
}
```

Создание экземпляра класса

```
ArrayAlgorithms aa =  
    new ArrayAlgorithms();  
aa.array = new int[50];  
...  
aa.sort(true);  
...  
int m = aa.max();
```

Создание экземпляра класса

```
ArrayAlgorithms aa = null;
```

```
aa = new ArrayAlgorithms();
```

```
aa = null;
```

Пакеты

```
package math.geometry;
```

```
class Line {  
    double A, B, C;  
    Line() { ... }  
}
```


Пакеты

```
package graphics.paint;
```

```
class Line {  
    int x1, y1, x2, y2;  
    Line() { ... }  
}
```

Пакеты

```
package main.console;
```

```
...
```

```
Line a = new Line();
```

```
Line b = new Line();
```

Пакеты

```
package main.console;
```

```
...
```

```
math.geometry.Line a =
```

```
    new math.geometry.Line();
```

```
graphics.paint.Line b =
```

```
    new graphics.paint.Line();
```

Пакеты

```
package math.analysis;
```

```
import math.geometry.*;
```

```
class Parabola {  
    Line tangent(double x) {  
        // реализация  
    }  
}
```

Пакеты

```
package math.analysis;
```

```
import math.geometry.Line;
```

```
class Parabola {  
    Line tangent(double x) {  
        // реализация  
    }  
}
```

Имя класса

- **math.geometry.Line**

полное имя класса

- **Line**

краткое имя класса

Можно не импортировать

- Классы из того же пакета
- Классы из пакета **java.lang**

Наследование

```
class Aaa {  
    void aaa();  
}
```

```
class Bbb extends Aaa {  
    void bbb();  
}
```


Наследование

```
Ааа x = new Ааа();  
x.aaa();  
//x.bbb(); ОШИБКА
```

```
Bbb y = new Bbb();  
y.aaa();  
y.bbb();
```

Наследование

```
Ааа z = new Bbb();  
z.aaa();  
//z.bbb(); ОШИБКА
```

Области видимости

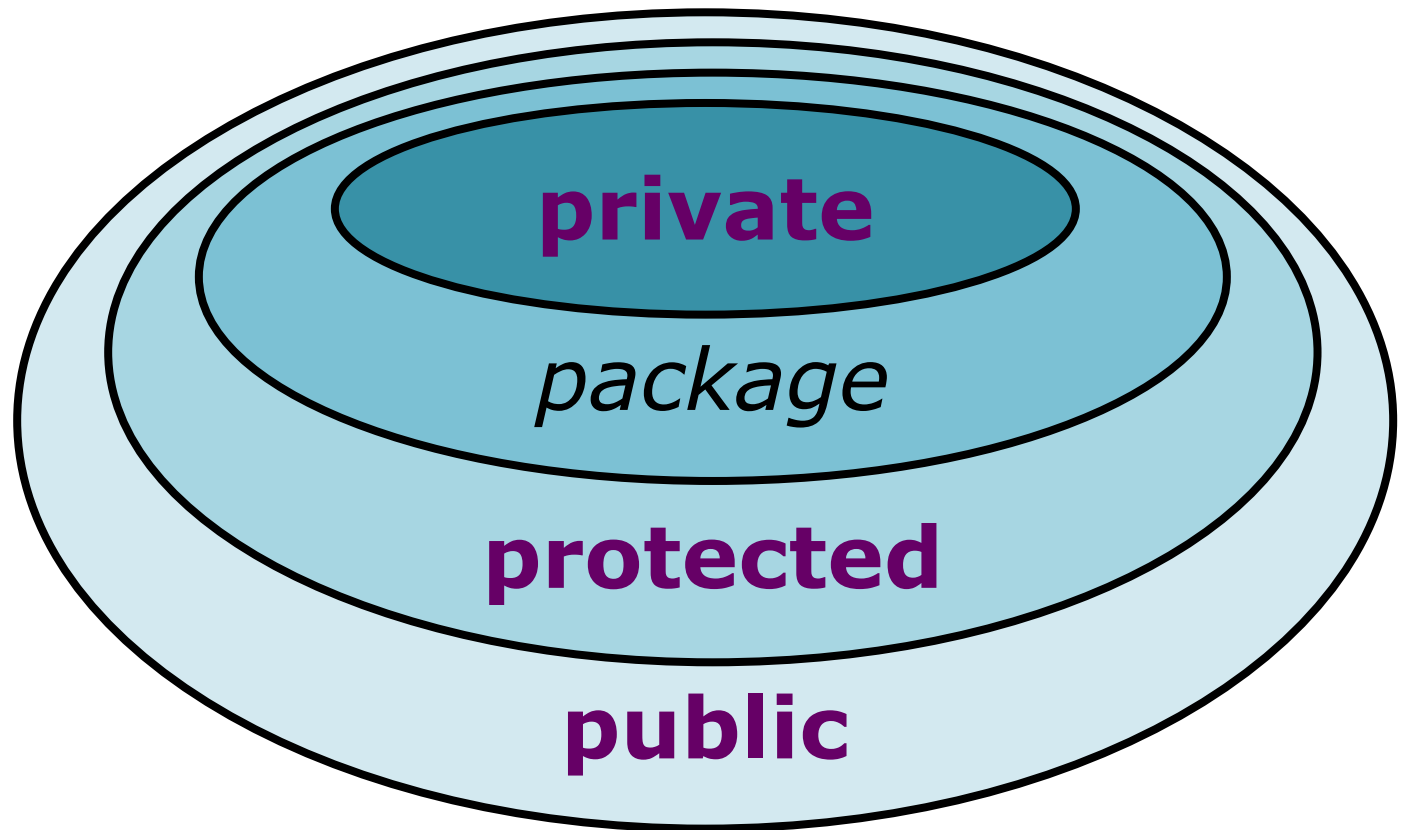
*// **виден везде***

```
public class ClassA {  
}
```

*// **виден только в своем пакете***

```
class ClassB {  
}
```

Области видимости



Примеры областей видимости

private int variable1;

void method1() { ... }

public double variable2;

String variable3;

protected void method2() { ... }

protected boolean variable4;

Модификатор **final**

- Класс
- Поле
- Метод

Модификатор **final**

```
final class TestA {  
}
```

// ОШИБКА

```
class TestB extends TestA {  
}
```

Модификатор **final**

```
class TestA {  
    void methodA() { /* 1 */ }  
    final void methodB() { /* 2 */ }  
}  
class TestB extends TestA {  
    void methodA() { /* 1 */ }  
    // ОШИБКА  
    void methodB() { /* 2 */ }  
}
```


Модификатор **final**

```
class TestA {  
    final int X = 10;  
}
```

Модификатор **static**

```
class Test {  
    int x;  
    static int y;  
    void methodA() {}  
    static void methodB() {}  
}
```

Модификатор **static**

```
Test a = new Test();  
a.x = 10;    a.y = 20;  
a.methodA(); a.methodB();
```

```
Test.y = 20;  
Test.methodB();
```

Модификатор **static**

```
Test a = new Test();
```

```
Test b = new Test();
```

```
a.y = 20;
```

```
System.out.println(b.y); // 20
```

Модификатор **static**

```
Test a = new Test();  
a.y = 20;
```

```
Test b = new Test();  
System.out.println(b.y); // 20
```

Модификатор **static**

```
Test.y = 20;
```

```
Test a = new Test();
```

```
System.out.println(a.y); // 20
```

```
Test b = new Test();
```

```
System.out.println(b.y); // 20
```

Модификатор **static**

```
Test a = new Test();
```

```
a.x = 10;      a.y = 20;
```

```
a.methodA();      a.methodB();
```

```
Test.y = 20;
```

```
Test.methodB();
```

Вызов методов суперкласса

```
class TestA {  
    void run() { /*1*/ }  
}  
class TestB extends TestA {  
    void run() { /*2*/ }  
    void test() {  
        run();  
        this.run();  
        super.run();  
    }  
}
```


Конструкторы

```
class Test {  
    Test() {  
        ...  
    }  
}
```

Конструкторы

```
class Test {  
    Test(int x) {  
        ...  
    }  
}
```

Конструкторы

```
class Test {  
    Test(int x) {  
        ...  
    }  
    Test() {  
        ...  
    }  
}
```

Конструкторы

```
class SubTest extends Test {  
    SubTest() {  
        super();  
    }  
}
```

Конструкторы

```
class SubTest extends Test {  
    SubTest() {  
        super(34);  
    }  
}
```

Конструкторы

```
class Test {  
    Test(int x) {  
        ...  
    }  
    Test() {  
        this(34);  
    }  
}
```

Класс **Object**

- Базовый для всех классов в Java
- Неявно является родителем класса, даже при отсутствии ключевого слова **extends**
- Содержит несколько стандартных методов

Класс **Object**

```
public class Man {  
    private String surname, name,  
                patronymic;  
    public Man(String surname,  
                String name,  
                String patronymic) {  
        this.surname = surname;  
        this.name = name;  
        this.patronymic = patronymic;  
    }  
}
```


Класс **Object**

```
public String toString() {  
    return surname + ' ' +  
    name.charAt(0) + '.' +  
    patronymic.charAt(0) + '.';  
}  
}
```

Класс **Object**

```
Man man = new Man("Пушкин",  
                  "Александр",  
                  "Сергеевич");
```

```
System.out.println(man);
```

```
// Пушкин А.С.
```

Класс **Object**

```
public class Man {  
    private String surname, name,  
                patronymic;  
    public Man(String surname,  
                String name,  
                String patronymic) {  
        this.surname = surname;  
        this.name = name;  
        this.patronymic = patronymic;  
    }  
}
```

Класс **Object**

```
public boolean equals(Object a) {  
    if(a instanceof Man) {  
        Man m = (Man) a;  
        return surname.equals(  
            m.surname);  
    }  
    return false;  
}  
}
```

Класс **Object**

```
public boolean equals(Object a) {  
    if(a instanceof Man) {  
        Man m = (Man) a;  
        return name.equals(m.name) &&  
            surname.equals(m.surname);  
    }  
    return false;  
}
```

Класс **Object**

```
private int age;  
public boolean equals(Object a) {  
    if(a instanceof Man) {  
        Man m = (Man) a;  
        return name.equals(m.name) &&  
            age == m.age;  
    }  
    return false;  
}
```

Класс **Object**

```
public int hashCode() {  
    /* реализация */  
}
```

Пример полиморфизма

```
class Stone {  
    double weight;  
    double price;  
    double cost() {  
        return price * weight;  
    }  
}
```


Пример полиморфизма

```
class Jewel extends Stone {  
    double transparent;  
    double cost() {  
        return super.cost()  
            * transparent;  
    }  
}
```

Пример полиморфизма

```
class RareJewel extends Jewel {  
    double cost() {  
        return super.cost()  
            * Math.log(Math.E  
                + weight);  
    }  
}
```

Пример полиморфизма

```
static Stone generate() {  
  int t = (int)(3*Math.random());  
  switch(t) {  
    case 0:  
      return new Stone();  
    case 1:  
      return new Jewel();  
    case 2:  
      return new RareJewel();  
  }  
}
```

Пример полиморфизма

```
Stone[] s = new Stone[25];  
for(int i = 0; i < s.length; i++){  
    s[i] = generate();  
}
```

```
for(...) {  
    if(s[i].cost() > s[j].cost())  
}
```

Пример абстрактного класса

```
public class Circle {  
    private double radius;  
    public double area() {  
        return Math.PI * radius  
            * radius;  
    }  
}
```

Пример абстрактного класса

```
public class Rectangle {  
    private double width;  
    private double height;  
    public double area() {  
        return width * height;  
    }  
}
```

Пример абстрактного класса

```
public class Rectangle  
    extends Figure {  
    public double area() { /*...*/ }  
}
```

```
public class Circle  
    extends Figure {  
    public double area() { /*...*/ }  
}
```

Пример абстрактного класса

```
public class Figure {  
    public double area() {  
        /* ??? */  
    }  
}
```


Пример абстрактного класса

```
public class Figure {  
    abstract public double area();  
}
```

Пример абстрактного класса

```
abstract public class Figure {  
    abstract public double area();  
}
```

Абстрактные методы и классы

```
abstract public class A {  
    abstract public void aaa();  
    abstract public void bbb();  
}
```

Абстрактные методы и классы

```
abstract public class B  
    extends A {  
    abstract public void aaa();  
    public void bbb() {  
        /* реализация */  
    }  
}
```

Абстрактные методы и классы

```
public class C extends B {  
    public void ааа() {  
        /* реализация */  
    }  
}
```

Абстрактные методы и классы

```
A x = new A(); // ОШИБКА  
B y = new B(); // ОШИБКА  
C z = new C();
```

Абстрактные методы и классы

```
abstract public class A {  
    public void aaa() {  
        /* реализация */  
    }  
    public void bbb() {  
        /* реализация */  
    }  
}
```

Пример абстрактного класса **без** абстрактных методов

```
public class Author {  
    private String surname;  
    private String name;  
    private String patronymic;  
    private String degree;  
}
```


Пример абстрактного класса **без** абстрактных методов

```
public class Book {  
    private String title;  
    private Author author;  
    private Integer year;  
}
```

Пример абстрактного класса **без** абстрактных методов

```
public class Reader {  
    private String surname;  
    private String name;  
    private String patronymic;  
    private Long phone;  
    private String address;  
}
```

Пример абстрактного класса **без** абстрактных методов

```
public class Usage {  
    private Reader reader;  
    private Book book;  
    private Date issueDay;  
    private Date returnDay;  
}
```

Пример абстрактного класса **без** абстрактных методов

```
public class Person {  
    private String surname;  
    private String name;  
    private String patronymic;  
}
```

Пример абстрактного класса **без**
абстрактных методов

```
public class Author extends  
    Person {  
    private String degree;  
}
```

Пример абстрактного класса **без**
абстрактных методов

```
public class Reader extends  
    Person {  
    private Long phone;  
    private String address;  
}
```

Пример абстрактного класса **без** абстрактных методов

```
abstract public class Person {  
    private String surname;  
    private String name;  
    private String patronymic;  
}
```

Пример чтения данных

```
public class BookTxtReader {  
    public Book[] read() {  
        /* реализация */  
    }  
}
```


Пример чтения данных

```
public class BookDBReader {  
    public Book[] read() {  
        /* реализация */  
    }  
}
```

Пример чтения данных

```
abstract public class BookReader {  
    abstract public Book[] read();  
}
```

Пример чтения данных

```
abstract public class BookReader {  
    abstract public  
        Book[] readBooks();  
    abstract public  
        Author[] readAuthors();  
}
```

Пример чтения данных

```
abstract public class BookReader {  
    public Book[] read() {  
        /* readAuthors() */  
        /* readBooks() */  
    }  
    abstract public  
        Book[] readBooks();  
    abstract public  
        Author[] readAuthors();  
}
```

Пример чтения данных

```
abstract public class BookReader {  
    public final Book[] read() {  
        /* readAuthors() */  
        /* readBooks() */  
    }  
    abstract protected  
        Book[] readBooks();  
    abstract protected  
        Author[] readAuthors();  
}
```

Пример сортировки

```
public class ReaderSorter {  
    public Reader[] sort(  
        Reader[] readers  
    ) {  
        /* реализация */  
    }  
}
```

Пример сортировки

```
public class ReaderSorter {  
    public Reader[] sortByName(  
        Reader[] readers  
    ) {  
        /* реализация */  
    }  
    public Reader[] sortByAddress(  
        Reader[] readers  
    ) {  
        /* реализация */  
    }  
}
```

Пример сортировки

```
abstract public  
class ReaderComparator {  
    abstract public int compare(  
        Reader r1,  
        Reader r2  
    );  
}
```


Пример сортировки

```
public class ReaderSorter {  
    public Reader[] sort(  
        Reader[] readers,  
        ReaderComparator comparator  
    ) {  
        /* начало цикла */  
        if(comparator  
            .compare(reader[i],  
                reader[j]) > 0) {  
            /* обмен */  
        }  
        /* конец цикла */  
    }  
}
```

Пример сортировки

```
public class ReaderByNameComparator  
    extends ReaderComparator {  
    public int compare(Reader r1,  
                      Reader r2) {  
        /* реализация */  
    }  
}
```

Пример сортировки

```
public class ReaderByAddressComparator  
    extends ReaderComparator {  
    public int compare(Reader r1,  
                    Reader r2) {  
        /* реализация */  
    }  
}
```

Пример сортировки

```
Reader[] readers = /* создание массива */
```

```
ReaderSorter sorter = new ReaderSorter();
```

```
ReaderComparator comparator;
```

```
comparator = new ReaderByNameComparator();
```

```
sorter.sort(readers, comparator);
```

```
/* вывод массива readers */
```

```
comparator = new  
    ReaderByAddressComparator();
```

```
sorter.sort(readers, comparator);
```

```
/* вывод массива readers */
```

Пример сортировки

```
abstract public  
class ReaderComparator {  
  
    abstract public int compare(  
        Reader r1,  
        Reader r2  
    );  
  
}
```

Пример сортировки

```
public interface ReaderComparator {  
    int compare(Reader r1, Reader r2);  
}
```

Пример сортировки

```
public class ReaderByNameComparator  
    implements ReaderComparator {  
  
    public int compare(Reader r1,  
                      Reader r2) {  
        /* реализация */  
    }  
}
```

Пример сортировки

```
public class ReaderByAddressComparator  
    implements ReaderComparator {
```

```
    public int compare(Reader r1,  
                    Reader r2) {
```

```
        /* реализация */
```

```
    }
```

```
}
```


Интерфейсы

```
public interface A {
```

```
    int x; // public final static
```

```
    void test(); // abstract public
```

```
}
```

Интерфейсы

```
public interface A {  
    int x; // public final static  
    void test(); // abstract public  
}
```

```
public class B implements A {}
```

```
public class C extends X  
    implements A {}
```

```
public class D implements A1, A2, A3 {}
```