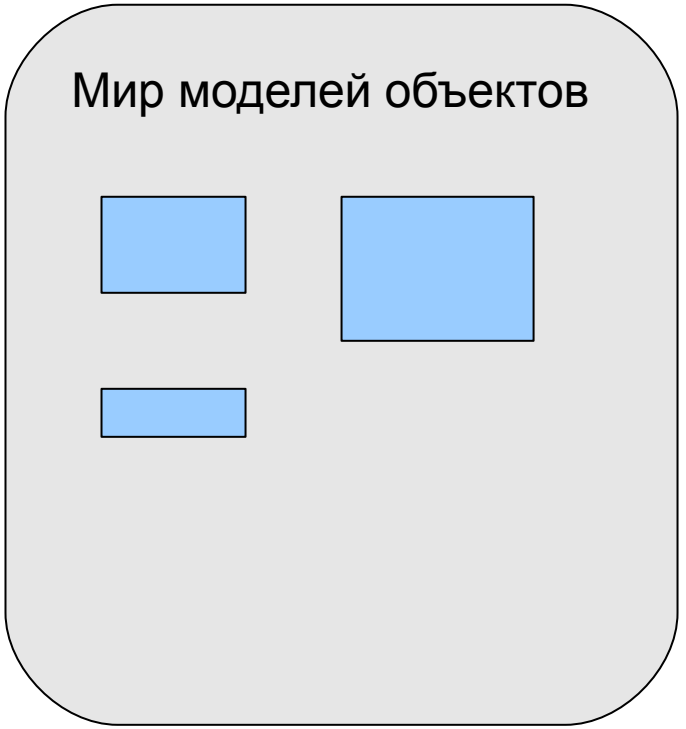
# Язык программирования Java.


## Примеры:

| | |
|---|---|
| Число 25 | Байт, содержащий последовательность битов “00011001” |
| Текст “Мой дядя...” | Область памяти, содержащая коды символов “М”, “о”,... |
| Самолет | Навигационные координаты точки<br>Набор эксплуатационных характеристик<br>Параметры загрузки на конкретный рейс |

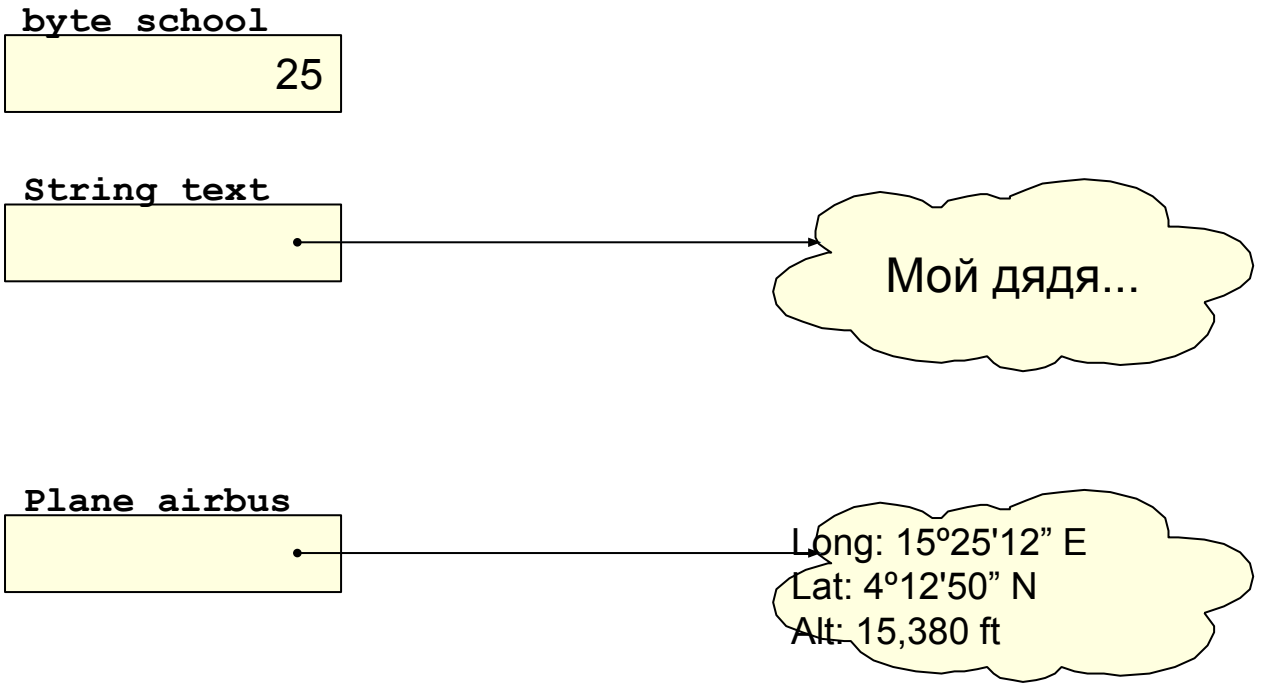
# Типы, значения, объекты, ссылки.

## Значения (объекты):

Занимают область памяти;
Имеют тип.

## Переменные (именованные участки памяти небольшого размера):

Занимают область памяти размером от 1 до 8 байтов;
Имеют имя и тип;
Могут содержать значение или ссылку на объект (в зависимости от типа).



# Переменные, присваивания.

Некоторые значения или объекты доступны всегда непосредственно. Они не меняются с течением времени (константы). В программе обозначаются “литералами”. Обычно имеют примитивный тип, но бывают и литералы для некоторых объектных типов (например, строки).

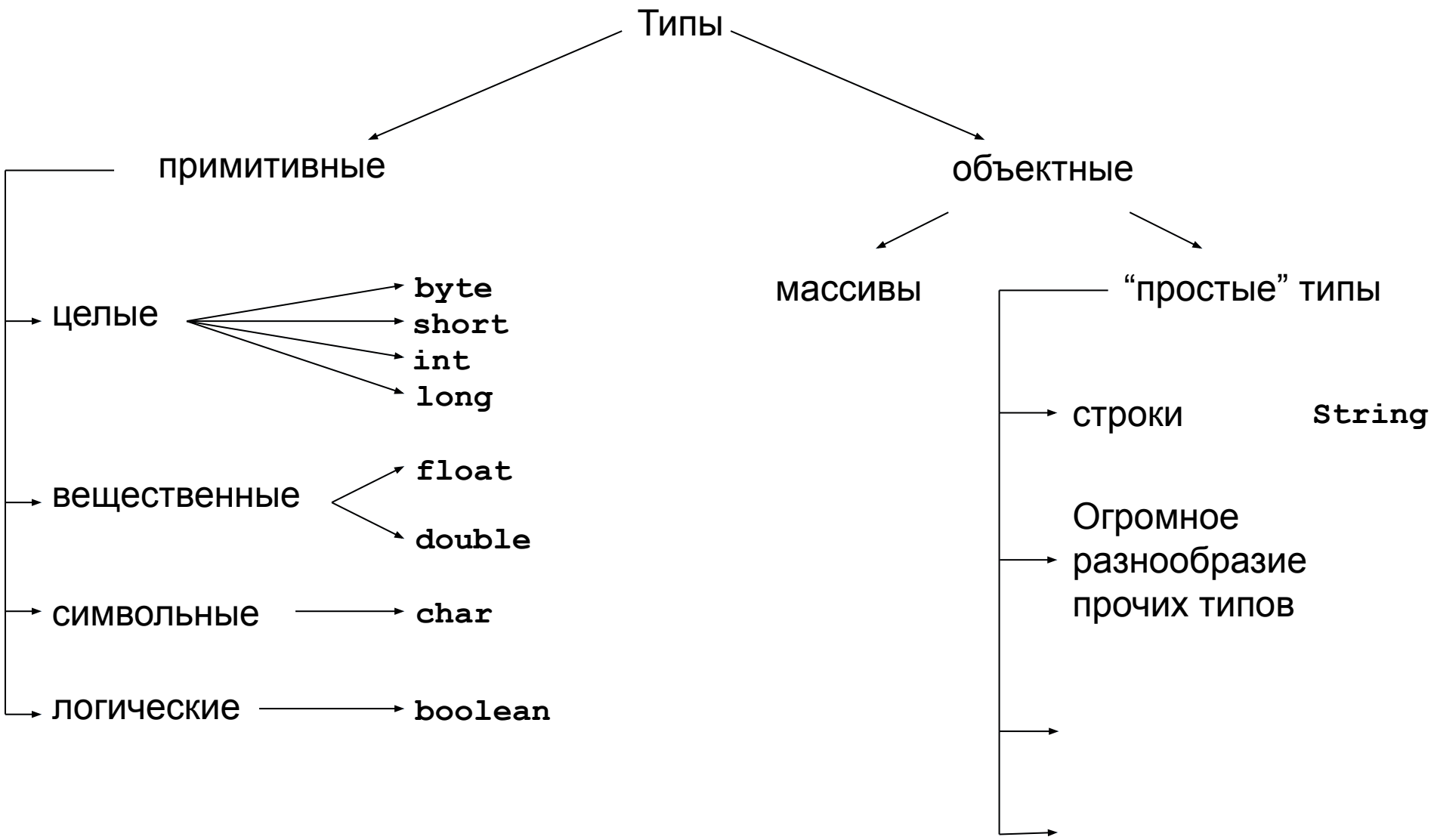
Для того, чтобы работать со значениями, надо:
1) “описать” переменную, задав ее тип;
2) “присвоить” ей значение.

```
byte school;        // Описание переменной типа byte с именем school.
String text;        // Описание переменной типа String с именем text.
Plane airbus;       // Описание переменной типа Plane с именем airbus.

school = 25;              // Присваивание константы-числа.
text = “Мой дядя...”;     // Присваивание константы-строки.
airbus = new Plane();     // Присваивание “вновь созданного” объекта типа Plane.

byte school = 25;               // Описание, совмещенное с присваиванием.
String text = “Мой дядя...”;    // Описание, совмещенное с присваиванием.
Plane airbus = new Plane();     // Описание, совмещенное с присваиванием.
```

# Типы данных. Иерархия типов.



# Простые программы.

## Операции, присваивание, управление.

С помощью операций можно получить новые значения из имеющихся или изменить значения имеющихся объектов.

С помощью присваиваний можно запомнить полученные значения для того, чтобы использовать их в дальнейшем.

С помощью управления можно задавать последовательность исполнения операций и присваиваний.

```
class TestProgram {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    n = 25 - n;
    System.out.format("Получили число %d\n", n);
  }
}
```

Операция **`Integer.parseInt`** задана в виде функции и преобразует строку в целое число.
Операция **`"-"`** задана в виде бинарного оператора и вычисляет новое значение по заданным.
Операция **`System.out.format`** задана в виде метода объекта **`System.out`**, формирует строку из заданных значений и других строк и выводит результат на консоль.

В переменной **`n`** последовательно запоминаются два разных значения.

Последовательное расположение операторов (“предложений”) программы задает линейное последовательное выполнение всех действий.

# Исполнение операций. Объектно-ориентированная и процедурная парадигмы.

Процедурная парадигма: операции и функции выполняются над пассивными операндами, в результате выполнения операции может появиться новый или измениться старый объект.

```
int n = Integer.parseInt("25");
n = 25 + n;
n += 25;
double sqrt2 = Math.sqrt(2.0);
```

Объектно-ориентированная парадигма: функции (методы) выполняются активным объектом, в результате исполнения метода может появиться новый или измениться существующий объект.

```
String s = "Hello, world".substring(3, 5);   // s = "lo";
System.out.println("Hello, world!");          // вывод строки на консоль.
```

Вызов методов записывается в “точечной нотации”. Однако, это лишь соглашение.

```
// Так писать НЕЛЬЗЯ, хотя по сути именно это и происходит в программе.
String s = String.substring("Hello, world", 3, 5);
System.println(System.out, "Hello, world!");

// Так писать тоже НЕЛЬЗЯ, но это хорошо соответствует ОО-стилю.
n = 25.add(n);
double sqrt2 = 2.0.sqrt();
```

# Присваивание. Преобразование типов.

Присваивание возможно только тогда, когда тип переменной совпадает с типом присваиваемого значения.

```
int n = "25";                  // НЕДОПУСТИМО!
n = Integer.parseInt("25");    // Возможно.
String s = 12;                 // НЕДОПУСТИМО!
s = String.valueOf(12);
s = "" + 12;    // В операции соединения строк происходит автоматическое
                // преобразование любого значения в строку.
```

Однако, возможно автоматическое “расширение” присваиваемого значения.

```
int i = 'a';
short s = 75;  // Константа 75 находится в допустимом диапазоне.
long lng = 10987654321L;  // Integer.MAX_VALUE = 2147483647;
                          // Long.MAX_VALUE = 9223372036854775807L;
lng = s;        // Автоматическое преобразование из short в long.
i = s + 12;     // Результат в операциях с целыми – всегда целый.
float f = lng + 1;
char b = 'a' + 1; // НЕВОЗМОЖНО! Надо: char b = (char)('a' + 1);
s = s + 12;       // НЕВОЗМОЖНО! Результат получается типа int.
                  // Надо: s += 12;  или  s = (short)(s + 12);
long lng = 10987654321L;  // Integer.MAX_VALUE = 2147483647;
                          // Long.MAX_VALUE = 9223372036854775807L;
i = f;            // НЕВОЗМОЖНО! Надо: i = (int)f.
i = lng - 12;     // НЕВОЗМОЖНО! Результат вычитания будет типа long.
int err = (int)10987654321L;  // Синтаксически допустимо, однако
                              // результат преобразования будет “странным”.
```

# Некоторые операции над целыми и вещественными значениями.

Операции с целыми операндами приводят к целому результату,
операции над вещественными – к вещественному.

```
17 / 3;     // = 5
17 % 3;     // = 2
17.0 / 3;   // = 5.666666666666667
8 % 2.5;    // = 0.5
3 << 4;     // = 48 (= 3 * 2^4)
51 >> 2;    // = 12 (= 51 / 2^2)
```

Поразрядные операции выполняются над битовыми (двоичными) представлениями
целых значений.

```
17 & 5;     // = 1  (10001 ^ 00101 = 00001)
17 | 5;     // = 21 (10001 | 00101 = 10101)
17 ^ 5;     // = 20 (10001 ^ 00101 = 10100)
~17;        // = -18 (~00...0010001 = 11...1101110)
```

Поразрядные операции полезны для работы с “битовыми шкалами”.
Например, чтобы инвертировать в шкале **n** бит номер **k** достаточно выполнить:

```
n ^= (1 << k);
```

а чтобы оставить в шкале **n** только младшие **k** разрядов, достаточно выполнить:

```
n &= (1 << k) - 1;
```