# Информационные технологии и программирование

Лекция 2

# Арифметические операторы

- Арифметические: +, –, *, /, %
– Деление работает как целочисленное, если оба аргумента целочисленные
- Унарные + и –.

# Сдвиги

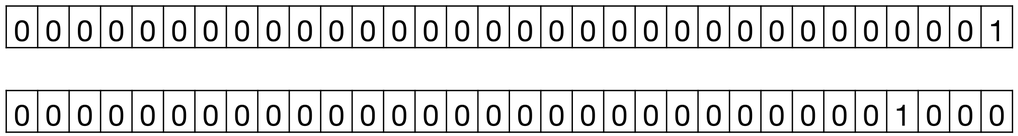
| | | |
|---|---|---|
| 5 | 0000 … 0000 0101 | |
| 5<<1 == 10 | 0000 … 0000 1010 | |
| 5>>1 == 2 | 0000 … 0000 0010 | |
| -5 | 1111 … 1111 1011 | |
| -5<<1 == -10 | 1111 … 1111 0110 | |
| -5>>1 == -3 | 1111 … 1111 1101 | старший (знаковый) бит сохраняется |
| -5>>>1 == 2147483645 | 0111 … 1111 1101 | старший (знаковый) бит заполняется нулем |

```
int a = 1<<35;
int b = 1<<3;
System.out.println (a==b);
```

# Сдвиги

| | | |
|---|---|---|
| 5 | 0000 … 0000 0101 | |
| 5<<1 == 10 | 0000 … 0000 1010 | |
| 5>>1 == 2 | 0000 … 0000 0010 | |
| -5 | 1111 … 1111 1011 | |
| -5<<1 == -10 | 1111 … 1111 0110 | |
| -5>>1 == -3 | 1111 … 1111 1101 | старший (знаковый) бит сохраняется |
| -5>>>1 == 2147483645 | 0111 … 1111 1101 | старший (знаковый) бит заполняется нулем |

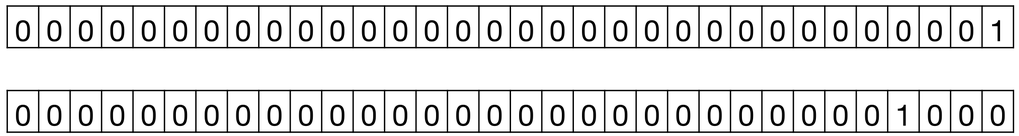
```
int a = 1<<35;
int b = 1<<3;
System.out.println (a==b);
```



# Сдвиги

| | | |
|---|---|---|
| 5 | 0000 … 0000 0101 | |
| 5<<1 == 10 | 0000 … 0000 1010 | |
| 5>>1 == 2 | 0000 … 0000 0010 | |
| -5 | 1111 … 1111 1011 | |
| -5<<1 == -10 | 1111 … 1111 0110 | |
| -5>>1 == -3 | 1111 … 1111 1101 | старший (знаковый) бит сохраняется |
| -5>>>1 == 2147483645 | 0111 … 1111 1101 | старший (знаковый) бит заполняется нулем |

```
int a = 1<<35;
int b = 1<<3;
System.out.println (a==b);
```

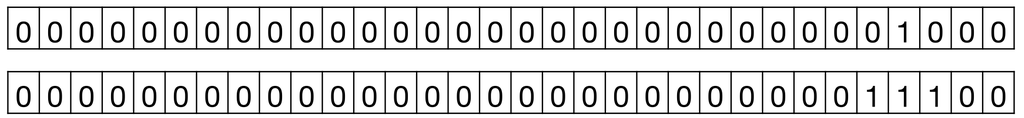



```
System.out.println (1<<32);
```

# Побитовые операторы

&, |, ^, ~

```
(1 == (n & 8) / 8)
```



# Операторы сравнения

- < <= > >= instanceof
- == !=

# Булевские операторы

- Без короткого замыкания: &, |, ^, ! (вместо тильды — восклицательный знак!)
- С коротким замыканием &&, ||.

```
x != 0 && 1 / x > x + y // no division by 0
```

# Пре/пост инкремент/декремент, присвоение с изменением

- a++, ++a
- a--, --a
- +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=

```
int m = 7;
int n = 7;
int a = 2 * ++m; // теперь значение a равно 16, а m равно 8
int b = 2 * n++; // теперь значение b равно 14, а n равно 8
```

# Тернарный оператор

x < y ? x :y

# Операторы в порядке убывания приоритета

| | | |
|---|---|---|
| Постфикс | () [] . (вызов метода) | Слева направо |
| Унарный | ++ -- ! ~ (приведение типов) new | Справа налево |
| Мультипликативный | * / % | Слева направо |
| Аддитивный | + - | Слева направо |
| Сдвиг | >> >>> << | Слева направо |
| Реляционный | > >= < <= instanceof | Слева направо |
| Равенство | == != | Слева направо |

# Операторы в порядке убывания приоритета

| | | |
|---|---|---|
| Побитовое И (AND) | & | Слева направо |
| Исключающее ИЛИ (XOR) | ^ | Слева направо |
| Побитовое ИЛИ (OR) | \| | Слева направо |
| Логическое И (AND) | && | Слева направо |
| Логическое ИЛИ (OR) | \|\| | Слева направо |
| Условный | ?: | Справа налево |
| Присваивание | = += -= *= /= %= >>= <<= &= ^= \|= | Справа налево |

```
a + = b + = c; // a + = (b + = c).
```

# Конструкция if

```
if (yourSales >= target)
    performance = "Satisfactory";
```

# If + block

```
if (yourSales >= target) {
    performance = "Satisfactory";
    bonus = 100;
}
```

# If + block + else

```
if (yourSales >= target) {
    performance = "Satisfactory";
    bonus = 100 + 0.01 * (yourSales - target);
}
else {
    performance = «Unsatisfactory";
    bonus = 0;
}
```

# else группируется с ближайшим if

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

# else группируется с ближайшим if

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

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

# Цепочки else if

```
if (yourSales >= 2 * target) {
    performance = "Excellent ";
    bonus = 1000;
}
else if (yourSales >= 1.5 * target ) {
    performance = "Fine";
    bonus = 500;
}
else if (yourSales >= target) {
    performance = "Satisfactory";
    bonus = 100;
}
else {
    System.out.println("You're fired");
}
```

# Цикл while

```
while (balance<goal){
    balance += payment;
    double interest = balance * interestRate / 100;
    balance += interest;
    years++;
}
System.out.println (years + " years.");
```

```
final boolean flag = false;
…
while (flag) {
. . . //не скомпилируется, unreachable code }
```

# do while

```
do {
    balance += payment;
    double interest = balance * interestRate / 100;
    balance += interest;
    year++;
    // print current balance
    ...
    // ask if ready to retire and get input
    ...
} while (input.equals("N"));
```

# for

```
for (int i = 1; i <= 10; i++)
    System.out.println(i);
```

И даже, хотя оператора «запятая» нет, можно так (но не нужно):

```
for (int i = 1; i <= 10; i++, j++)
    System.out.println(i);
```

# Видимость переменной цикла

```
for (int i = 1; i <= 10; i++) {
    ...
}
// i no longer defined here
int i;
for (i = 1; i <= 10; i++) {
    ...
}
// i is still defined here
```

# Использование break и continue

```
while (years <= 100) {
    balance += payment;
    double interest = balance * interestRate / 100;
    balance += interest;
    if (balance >= goal)
       break;
    years++;
}
```

Переход к следующему циклу:

```
Scanner in = new Scanner(System.in);
while (sum < goal) {
    System.out.print("Enter a number: ");
    n = in.nextInt();
    if (n < 0)
        continue;
    sum += n; // not executed if n < 0
}
```

# Метки для break и continue

```
Scanner in = new Scanner(System.in);
int n;
// label is here!
read_data: while (. . .) {
    ...
    for (. . .) {
        System.out.print("Enter a number >= 0: ");
        n = in.nextInt();
        if (n < 0)
            break read_data;
            // break out of read_data loop
        ...
    }
}
if (n < 0) {
    // deal with bad situation
}
else {
    // carry out normal processing
}
```

# switch

```
Scanner in = new Scanner(System.in);
System.out.print("Select an option (1, 2, 3, 4) ");
int choice = in.nextInt();
switch (choice){
    case 1:
        ...
        break;
    case 2:
        ...
        break;
    case 3:
        ...
        break;
    case 4:
        ...
        break;
    default:
        //неверный ввод
        break;
}
```

# switch-case особенности

- Не забываем `break`-и (утилиты типа Checkstyle напоминают), иначе выполняем всё до конца `switch`, как в C!
- switch бывает: по целому, по `char`-у, по `String`-у (Java 7+) и по `enum`-у.

# Массивы

- Из любого типа можно построить массив данного типа.
- Длина массива может быть определена в runtime, но после создания не может быть изменена.
- Массив аллоцируется в куче и передаётся по ссылке.
- Массивы проверяют тип данных (`ArrayStoreException`) и границы (`ArrayIndexOutOfBoundsException`) в run-time.
- Правда жизни: скорее всего, вы не будете использовать массивы в современном коде.

# Декларирование и инициализация массива

Два варианта:

```
int[] a
int a[]  — не делайте так
```

Инициализация:

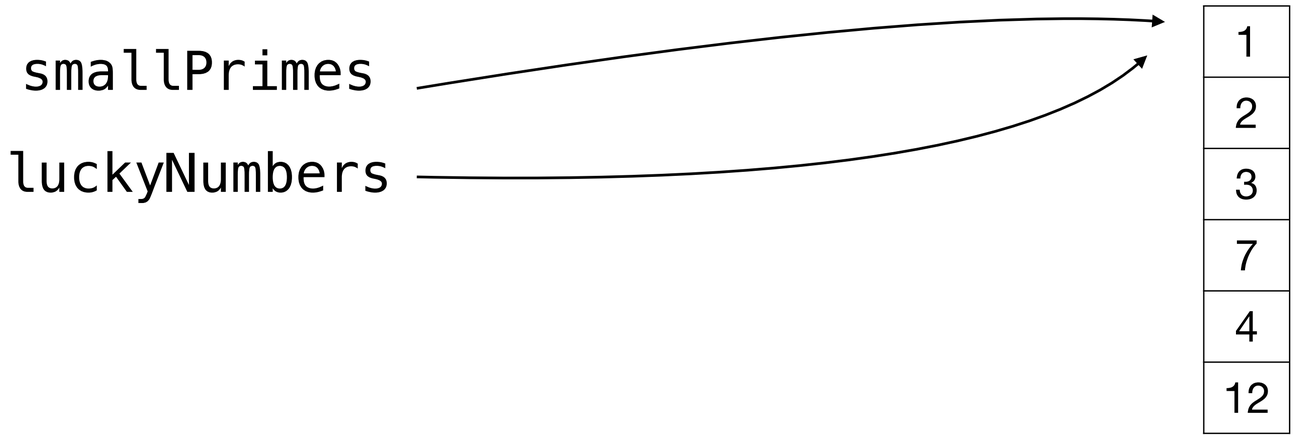
```
int[] a = new int[100];
int[] a = {1, 3, 5};
```

анонимный массив:

```
foo(new int[] {2, 4, 6});
```

# Массивы передаются по ссылке

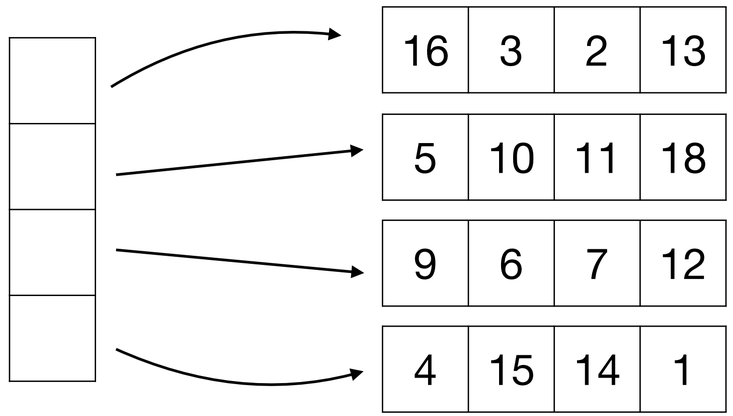
```
int[] luckyNumbers = smallPrimes;
luckyNumbers[5] = 12; // теперь smallPrimes[5] тоже 12
luckyNumbers = Arrays.copyOf(luckyNumbers, 2 * luckyNumbers.length);
//теперь luckyNumbers это отдельный массив
//и он стал в два раза длиннее
```



# Многомерные массивы (на самом деле их нет)

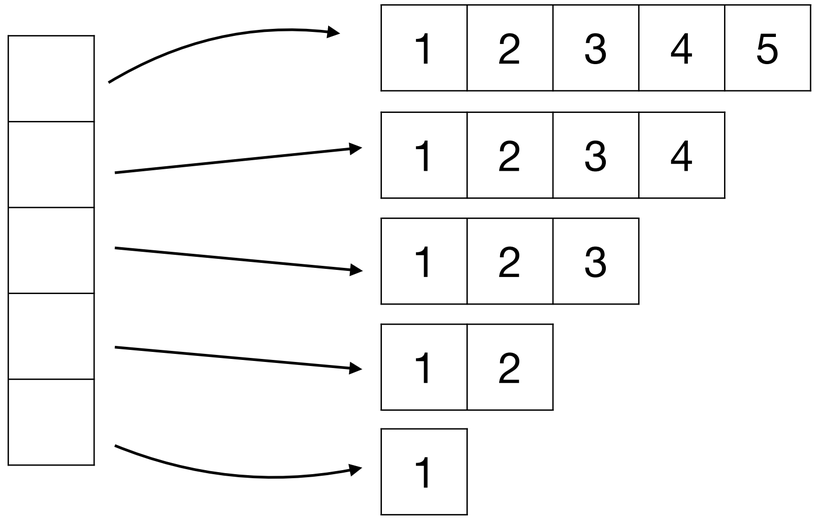
Есть массивы массивов

```
int[][] magicSquare = {
    {16, 3, 2, 13},
    {5, 10, 11, 18},
    {9, 6, 7, 12},
    {4, 15, 14, 1}
};
```



# Треугольные матрицы

```
//объявляем и создаём массив, указывая только количество строк
int [][] twoDimArray = new int[5][];

//инициализируем массив, заполняя его массивами разной длины
twoDimArray[0] = new int[]{1, 2, 3, 4, 5};
twoDimArray[1] = new int[]{1,2,3,4};
twoDimArray[2] = new int[]{1,2,3};
twoDimArray[3] = new int[]{1,2};
twoDimArray[4] = new int[]{1};
```



# Все есть класс

- Любой код - метод некого класса
- Любые данные хранятся в полях некоторого класса
- Любые типы данных (исключая примитивные, но включая массивы) - наследники класса Object

# Классы помещаются в пакеты

- `ru.mtuci.example`
- `ru.mtuci.example.name`
- Каждый .java-файл начинается с объявления пакета: `package ru.mtuci.hello`
- В корне пакета может быть `package-info.java`, не содержащий классы, а только JavaDoc над ключевым словом `package`
- <Имя пакета>.<имя класса> задаёт полный идентификатор любого класса, доступного в исходном коде или через библиотеки (например, `ru.mtuci.hello.App`)
- Вложенные пакеты - это разные пакеты с точки зрения Java (package-private одного пакета не будут видны в другом)

# Структура класса: поля, конструкторы, методы

```
class ClassName
{
    field1
    field2
    . . .
    constructor1
    constructor2
    . . .
    method1
    method2
    . . .
}
```

# Описание класса

```
package org.megacompany.staff;
class Employee {
    // instance fields
    private String name;
    private double salary;
    private LocalDate hireDay;
    // constructor
    public Employee(String n, double s, int year, int month, int day) {
        name = n;
        salary = s;
        hireDay = LocalDate.of(year, month, day);
    }
    // a method
    public String getName() {
        return name;
    }
    // more methods
    . . .
}
```

# Создаем и используем экземпляры класса

```
//При необходимости, импортируем
import org.megacompany.staff.Employee;

//где-то в теле метода
. . .
Employee hacker = new Employee("Harry Hacker", 50000, 1989, 10, 1);
Employee tester = new Employee("Tommy Tester", 40000, 1990, 3, 15);

hacker.getName(); //returns "Harry Hacker"
```

# Инициализация полей

- В отличие от локальных переменных, поля можно не инициализировать явно.
- В этом случае примитивные типы получают значение по умолчанию (`0`, `false`), а поля со ссылками — значение `null`.
- Проинициализировать поле по месту его определения не возбраняется:
  `int a = 42` или даже `int a = getValue()`.

# Поле this

```
{
    ...

    int value;

    setValue(int value) {
        //поле перекрыто аргументом
        this.value = value;
    }

    registerMe(Registrator r) {
        //нужна ссылка на себя
        r.register(this);
    }
}
```

# Объект передается по ссылке

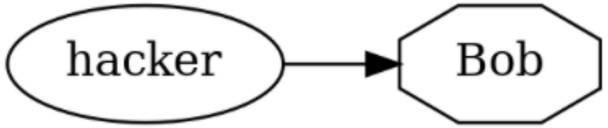
```
public class Employee {
    int age = 18;

    public static void main(String[] args) {
        Employee e = new Employee();
        int a = 1;
        foo(e, a);
        System.out.printf("%d - %d", e.age, a);
        //prints 42 - 1
    }

    static void foo(Employee e, int a) {
        //e passed by reference, a passed by value
        e.age = 42;
        a = 5;
    }
}
```

# Рождение, жизнь и смерть объекта

```
new Employee("Bob")
```

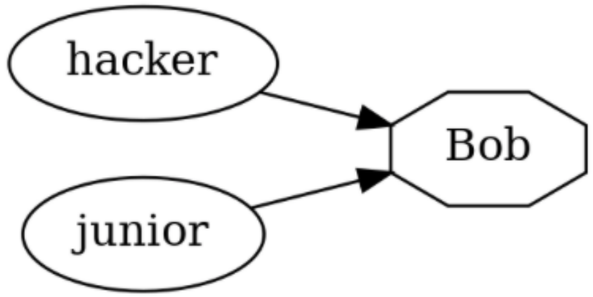
# Присваивание ссылки

```
Employee hacker = new Employee("Bob");
```



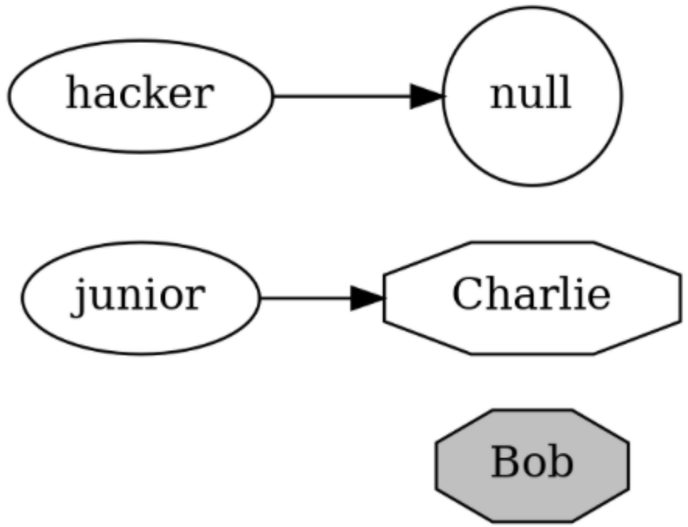
# Присваивание ссылки

```
Employee junior = hacker;
```



# Потеря ссылки

```
hacker = null;
junior = new Employee("Charlie");
```



# Сборка мусора

```
hacker = null;
junior = new Employee("Charlie");
```