Laboratory Work #03

Java Basic Syntax. Java Primitive Data Types and Simple Operators (operations)

LEARN. GROW. SUCCEED.

® 2020-2021. Department: <Software of Information Systems and Technologies>
Faculty of Information Technology and Robotics
Belarusian National Technical University
by Viktor Ivanchenko / ivanvikvik@bntu.by / Minsk

ЛАБОРАТОРНАЯ РАБОТА #03
Базовый синтаксис языка Java. Примитивные типы данных Java и простые операторы (операции)

Цель работы 

Ознакомиться с базовым синтаксисом и системой типов Java; изучить примитивные типы данных и простые операторы (операции).

Требования 

1) При разработке кода можно использовать любую интегрированную среду разработки. Однако, при запуске программы рекомендуется вручную задействовать основные компоненты Java (компилятор – ***javac***, утилиту для запуска JVM – ***java***).
2) При разработке программ придерживайтесь соглашений по написанию кода на JAVA (Java Code-Convention*)*.

Задание

Необходимо создать исследовательское приложение, которое тестирует все возможные (разрешённые) операции (*арифметические*, *операции отношения (сравнения)*, *логические*, *побитовые (бинарные)*, *операции составного присваивания и другие операции*) над разрешёнными соответствующими примитивными типами данных языка программирования Java (*byte, short, int, long, float, double, char, boolean*) с использованием статических методов класса.

Best of LUCK with it, and remember to HAVE FUN while you're learning :)
Victor Ivanchenko

Пример разработки исследовательской программы для тестирования разрешённых операций в Java над примитивными типами данных int и boolean

Ниже представлен примерный исходный код двух классов: исследовательского класса ***DataTypesTester*** и класса ***Lab03***, в котором есть стартовая точка входа в программу.

Класс *DataTypesTester* содержит методы, в которых описан код для тестирования всех разрешённых операций над соответствующими типами данных, в частности, метод ***testInt()*** тестирует все операции над типом int, метод тест ***testBool()*** тестирует все операции над типом boolean и т.д.

Класс ***Lab03*** содержит единственный метод ***main(..)***, который является стартовой точкой запуска и начало работы тестируемой программы. Данный метод предназначен для непосредственного вызова и выполнения методов ***testInt()***, ***testBoolean()*** и других.

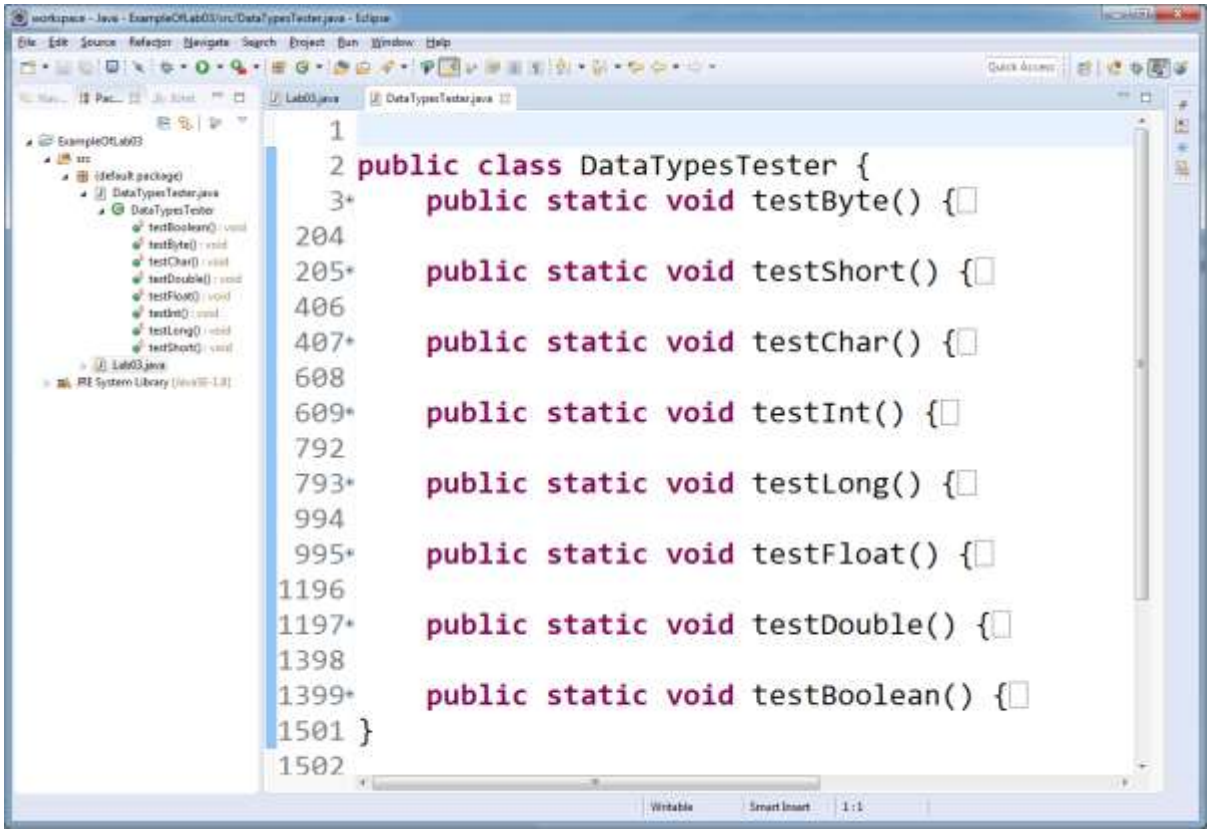


Рисунок 1 – Исходный код тестового класса *DataTypesTester*

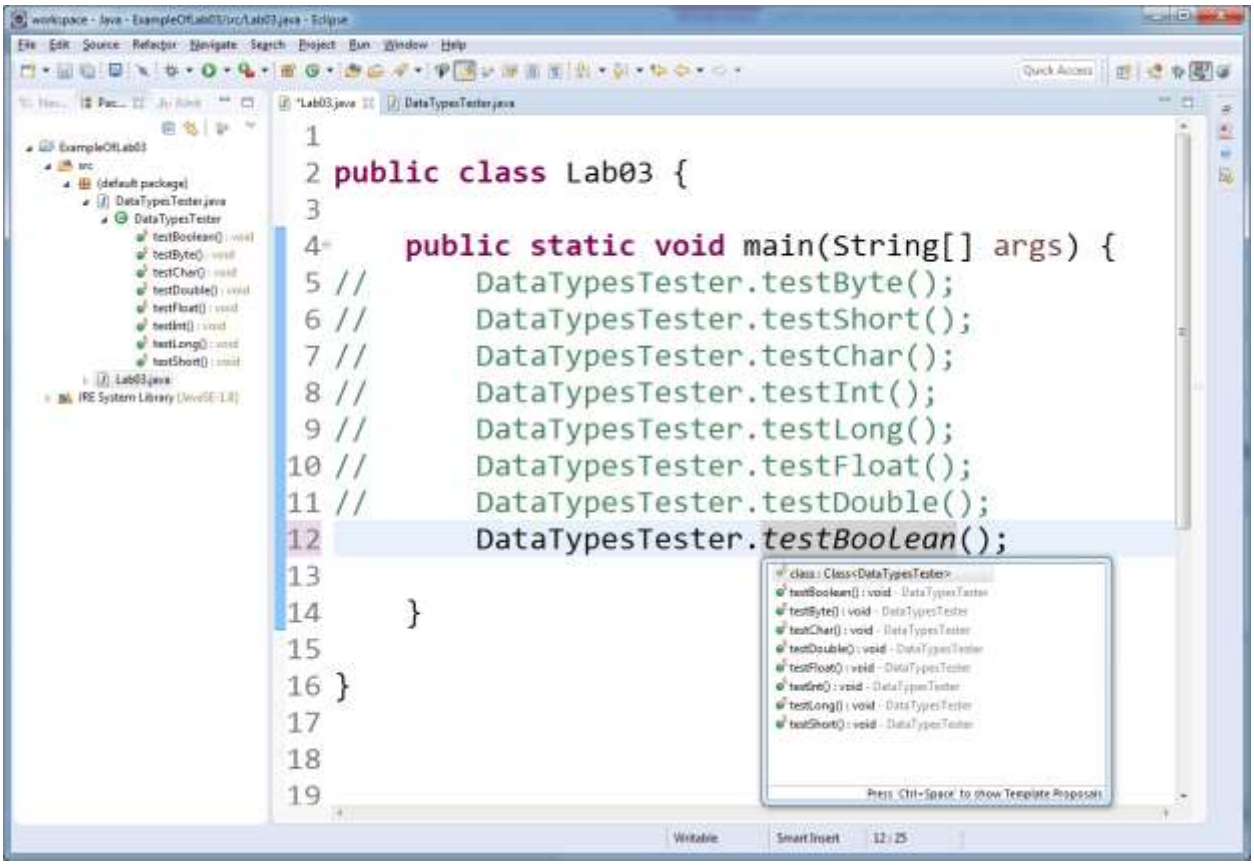


Рисунок 2 – Исходный код стартового класса *Lab03* программы

Ниже приведено примерное описание исходного кода метода ***testInt()*** и метода ***testBoolean()*** тестового класса ***DataTypesTester***:

```
public class DataTypesTester {

	public static void testByte() {…}
	public static void testShort(){…}
	public static void testChar(){…}

	public static void testInt() {
		int a = 5, b = 2, c;

		System.out.println("\n***** Arithmetical Operators *****");

		// addition
		c = a + b;
		System.out.printf("%d + %d = %d\n", a, b, c);
		// subtraction
		c = a - b;
		System.out.printf("%d - %d = %d\n", a, b, c);
		// multiplication
		c = a * b;
		System.out.printf("%d * %d = %d\n", a, b, c);
		// division
```

```
c = a / b;
System.out.printf("%d / %d = %d\n", a, b, c);
// modulus
c = a % b;
System.out.printf("%d %% %d = %d\n", a, b, c);

// c = a / 0; --> Arithmetical Exception
// c = a % 0; --> Arithmetical Exception

a = 5;
// unary minus
c = -a;
System.out.printf("-%d = %d\n", a, c);
// unary plus
c = +a;
System.out.printf("+%d = %d\n", a, c);
// prefix increment
a = 5;
System.out.printf("++%d = %d\n", a, ++a);
// postfix increment
a = 5;
System.out.printf("%d++ = %d\n", a, a++);
// prefix decrement
a = 5;
System.out.printf("--%d = %d\n", a, --a);
// postfix decrement
a = 5;
System.out.printf("%d-- = %d\n", a, a--);

System.out.println("\n***** Bitwise Operators *****");

c = a & b; // bitwise AND
System.out.printf("%d & %d = %d\n", a, b, c);
System.out.println("in binary form:");
System.out.printf("%s & %s = %s\n", Integer.toBinaryString(a),
            Integer.toBinaryString(b), Integer.toBinaryString(c));

c = a | b; // bitwise OR
System.out.printf("%d | %d = %d\n", a, b, c);
System.out.println("in binary form:");
System.out.printf("%s | %s = %s\n", Integer.toBinaryString(a),
            Integer.toBinaryString(b), Integer.toBinaryString(c));

c = a ^ b; // bitwise XOR
System.out.printf("%d ^ %d = %d\n", a, b, c);
System.out.println("in binary form:");
System.out.printf("%s ^ %s = %s\n", Integer.toBinaryString(a),
            Integer.toBinaryString(b), Integer.toBinaryString(c));

c = ~a; // bitwise unary compliment
System.out.printf("~%d = %d\n", a, c);
System.out.println("in binary form:");
System.out.printf("~%s = %s\n", Integer.toBinaryString(a),
            Integer.toBinaryString(c));
c = a << 1; // left shift
System.out.printf("%d << 1 = %d\n", a, c);
```

Обратите внимание, как выводиться знак процент в строке.

Запомните, что при использовании операций деления и остатка от деления над целыми числами может произойти исключительная ситуация ***ArithmeticalException*** во время работы программы!!!

```
System.out.println("in binary form:");
System.out.printf("%s << 1 = %s\n", Integer.toBinaryString(a),
            Integer.toBinaryString(c));

c = a >> 1; // right shift
System.out.printf("%d >> 1 = %d\n", a, c);
System.out.println("in binary form:");
System.out.printf("%s >> 1 = %s\n", Integer.toBinaryString(a),
            Integer.toBinaryString(c));

a = -5;
c = a >> 1; // right shift
System.out.printf("%d >> 1 = %d\n", a, c);
System.out.println("in binary form:");
System.out.printf("%s >> 1 = %s\n", Integer.toBinaryString(a),
            Integer.toBinaryString(c));

a = 5;
c = a >>> 1; // zero fill right shift
System.out.printf("%d >>> 1 = %d\n", a, c);
System.out.println("in binary form:");
System.out.printf("%s >>> 1 = %s\n", Integer.toBinaryString(a),
            Integer.toBinaryString(c));

a = -5;
c = a >>> 1; // zero fill right shift
System.out.printf("%d >>> 1 = %d\n", a, c);
System.out.println("in binary form:");
System.out.printf("%s >>> 1 = %s\n", Integer.toBinaryString(a),
            Integer.toBinaryString(c));

System.out.println("\n***** Assignment Operators *****");

c = 7;
System.out.printf("%d += %d --> c = %d\n", c, a, c += a);
System.out.printf("%d -= %d --> c = %d\n", c, a, c -= a);
System.out.printf("%d *= %d --> c = %d\n", c, a, c *= a);
System.out.printf("%d /= %d --> c = %d\n", c, a, c /= a);
System.out.printf("%d %%= %d --> c = %d\n", c, a, c %= a);

// c %= 0; --> Arithmetical Exception
// c /= 0; --> Arithmetical Exception

System.out.printf("%d |= %d --> c = %d\n", c, a, c |= a);
System.out.printf("%d &= %d --> c = %d\n", c, a, c &= a);
System.out.printf("%d ^= %d --> c = %d\n", c, a, c ^= a);
System.out.printf("%d >>= 1 --> c = %d\n", c, c >>= 1);
System.out.printf("%d <<= 1 --> c = %d\n", c, c <<= 1);
System.out.printf("%d >>>= 1 --> c = %d\n", c, c >>>= 1);

System.out.println("\n***** Relational Operators *****");

// greater than
System.out.printf("%d > %d --> %b\n", a, b, a > b);
// greater than or equal to
System.out.printf("%d >= %d --> %b\n", a, b, a >= b);
```

```
// less than
System.out.printf("%d < %d --> %b\n", a, b, a < b);
// less than or equal to
System.out.printf("%d <= %d --> %b\n", a, b, a <= b);
// equal to
System.out.printf("%d == %d --> %b\n", a, b, a == b);
// not equal to
System.out.printf("%d != %d --> %b\n", a, b, a != b);

System.out.println("\n***** Logical Operations *****");

// logical AND
System.out.printf("(%d > %d) && (%d > 0) --> %b\n", a, b, a,
            (a > b) && (a > 0));

// bitwise logical AND
System.out.printf("(%d > %d) & (%d > 0) --> %b\n", a, b, a,
            (a > b) & (a > 0));

// logical OR
System.out.printf("(%d >= %d) || (%d != 0) --> %b\n", a, b, b,
            (a >= b) || (b != 0));

// bitwise logical OR
System.out.printf("(%d >= %d) | (%d != 0) --> %b\n", a, b, b,
            (a >= b) | (b != 0));

// logical XOR
System.out.printf("(%d >= %d) | (%d != 0) --> %b\n", a, b, b,
            (a >= b) ^ (b != 0));

// logical NOT
System.out.printf("!(%d >= %d) --> %b\n", a, b, !(a >= b));

System.out.println("\n***** Misc Operators *****");

System.out.println("\nCondition Operator:");

System.out.printf("%d > %d ? %d : %d --> %d\n", a, b, a, b,
            (a > b ? a : b));

System.out.println("\nType Cast Operator:");

byte bt = 1;
short sh = -32000;
char ch = '\u0002';
long l = 100000000000000000L;
float f = 1.9f;
double d = 123456789.625;
boolean bool = true;

c = bt;
System.out.printf("int = byte: c = %d --> c = %d\n", bt, c);
c = sh;
```

```
        System.out.printf("int = short: c = %d --> c = %d\n", sh, c);
        c = ch;
        System.out.printf("int = char: c = '%c' --> c = %d\n", ch, c);
        c = (int) l;
        System.out.printf("int = long: c = (int)%d --> c = %d\n", l, c);
        c = (int) f;
        System.out.printf("int = float: c = (int)%f --> c = %d\n", f, c);
        c = (int) d;
        System.out.printf("int = double: c = (int)%f --> c = %d\n", d, c);
        // c = (int)bool;
        System.out.printf("int = boolean: c = (int)%b --> Compile Error\n",
                    bool);
}

public static void testLong(){…}
public static void testDouble(){…}

public static void testBoolean() {
        boolean a = true, b = false, c;

        System.out.println("\n***** Arithmetical Operators *****");

        // c = a + b; // addition
        // c = a - b; // subtraction
        // c = a * b; // multiplication
        // c = a / b; // division
        // c = a % b; // modulus
        // c = -a; // unary minus
        // c = +a; // unary plus
        // c = ++a; // prefix increment
        // c = a++; // postfix increment
        // c = --a; // prefix decrement
        // c = a--; // postfix decrement

        System.out.println("\n***** Bitwise Operators *****");

        c = a & b; // bitwise AND
        System.out.printf("%b & %b = %b\n", a, b, c);

        c = a | b; // bitwise OR
        System.out.printf("%b | %b = %b\n", a, b, c);

        c = a ^ b; // bitwise XOR
        System.out.printf("%b ^ %b = %b\n", a, b, c);

        // c = ~a; // bitwise unary compliment
        // c = a << 1; // left shift
        // c = a >> 1; //right shift
        // c = a >>> 1; // zero fill right shift

        System.out.println("\n***** Assignment Operators *****");

        System.out.printf("%b |= %b --> c = %b\n", c, a, c |= a);
        System.out.printf("%b &= %b --> c = %b\n", c, a, c &= a);
        System.out.printf("%b ^= %b --> c = %b\n", c, a, c ^= a);
```

Над типом ***boolean*** не определён ни один математический оператор.

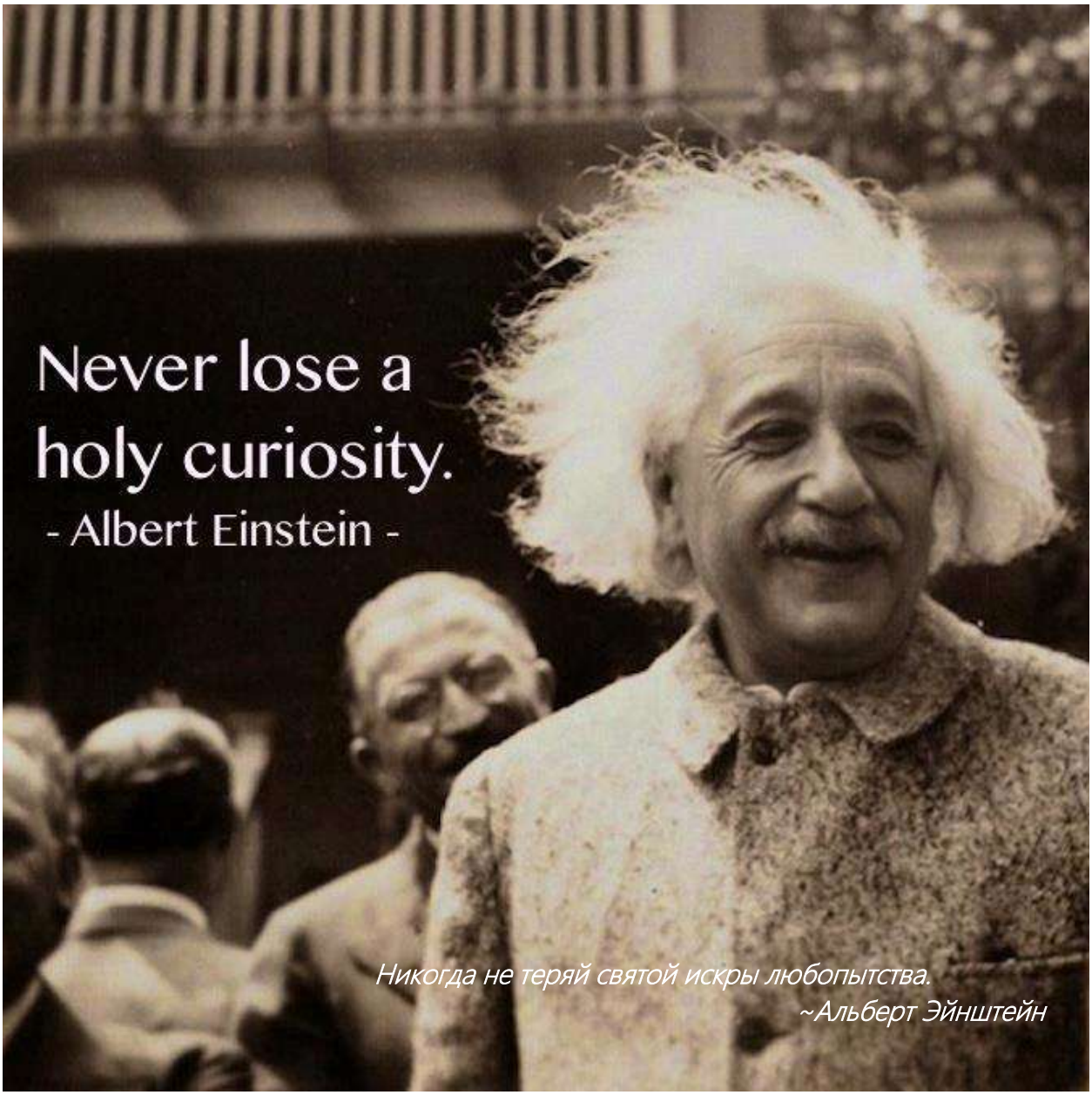
Над типом ***boolean*** определены соответствующие операции составного присваивания и часть побитовых операций, которые аналогичны по действию и результату соответствующим логическим операциям.

```
// c += a;
// c -= a;
// c *= a;
// c /= a;
// c %= a;
// c >>= 1;
// c <<= 1;
// c >>>= 1;

System.out.println("\n***** Relational Operators *****");

// equal to
System.out.printf("%b == %b --> %b\n", a, b, a == b);
// not equal to
System.out.printf("%b != %b --> %b\n", a, b, a != b);

// c = a > b; // greater than
// c = a >= b; // greater than or equal to
// c = a < b; // less than
// c = a <= b; // less than or equal to

System.out.println("\n***** Logical Operations *****");

c = a && b; // logical AND
System.out.printf("%b && %b = %b\n", a, b, c);

c = a || b; // logical OR
System.out.printf("%b || %b = %d\n", a, b, c);

c = a ^ b; // logical XOR
System.out.printf("%b ^ %b = %b\n", a, b, c);

// logical NOT
System.out.printf("!%b --> %b\n", a, !a);

System.out.println("\n***** Misc Operators *****");

System.out.println("\nCondition Operator:");

System.out.printf(" %b ? %b : %b --> %b\n", c, a, b, (c ? a : b));

System.out.println("\nType Cast Operator:");

// byte bt = 1;
// short sh = -32000;
// char ch = '\u0002';
// long l = 100000000000000000L;
// float f = 1.9f;
// double d = 123456789.625;
boolean bool = true;

// c = (boolean) bt;
// c = (boolean) sh;
// c = (boolean) ch;
// c = (boolean) l;
// c = (boolean) f;
```

Значения переменных типа ***boolean*** можно сравнивать только на равенство и неравенство.

Значение типа ***boolean*** нельзя преобразовать ни к какому типу, а также его нельзя получить ни из какого типа.

```
            // c = (boolean) d;
            c = bool;
        }
}
```



Контрольные вопросы 

1. Опишите систему типов в Java.
2. Каковы основные отличия примитивных типов данных от ссылочных?
3. Опишите базовые примитивные типы данных языка Java (***byte***, ***short***, ***char***, ***int***, ***long***, ***float***, ***double***, ***boolean***) и их характеристики.
4. Какие пять вещей характеризуют операции в языках программирования?
5. Какие существуют группы операций в Java и по какому признаку операции собираются в группы?
6. Опишите доступные операции в языке Java:
   - ***arithmetic operators*** (арифметические операции);
   - ***relational operators*** (операции отношения/сравнения);
   - ***logical operators*** (логические операции);
   - ***bitwise operators*** (побитовые/бинарные операции);
   - ***assignment operators*** (присваивание и операции составного присваивания);
   - ***misc operators*** (другие операции: ***instanceof*** – оператор проверки принадлежности типу, ***? :*** – условный оператор (*conditional operators*), ***(type)*** – оператор приведения типа (*type cast operator*), оператор ***new*** и др.).
7. Для каких типов данных у языка Java и JMV нет встроенных операций?
8. Как в языке Java осуществляет вычисление выражения, если в нём находятся данные различных типов?
9. Какие операции и над какими типами данных могут выкинуть исключительную ситуацию во время выполнения программы?
10. Есть ли в Java перегрузка операторов (операций)? Какие операторы (операции) всё же перегружены на уровне языка Java и для каких типов данных?
11. В чём существенная разница между операциями **||** (логическое ИЛИ) и **|** (побитовое ИЛИ), а также **&&** (логическое И) и **&** (побитовое И)?
12. Что такое **приоритет** выполнения операций?
13. Какие операции имеют наибольший приоритет в языке Java?
14. Какие операции имеют наименьший приоритет в языке Java?
15. Как распределён приоритет между префиксным и постфиксным инкрементом (декрементом)?
16. Как распределён приоритет между операциями логической группы?
17. Что такое **ассоциативность** выполнения операций?
18. Какие операции выполняются слева направо?

19. Какие операции выполняются справа налево?
20. Контролируется ли в Java переполнение (выход значения за границы выделенной памяти) при вычислении выражений?
21. Приведите схему и опишите, каким образом работает **сужающее** (≈ понижающее ≈ **явное**) и **расширяющее** (≈ повышающее ≈ разрешённое ≈ **неявное** ≈ автоматическое) приведение примитивных типов.
22. Опишите особенности потери точности при явном и неявном приведении типов.
23. При работе с какими операциями не требуется явное приведение типов?
24. Зачем в объектно-ориентированном языке Java поддерживаются примитивные типы данных?
25. Зачем в Java существуют *Wrapper-классы* (классы-обвёртки, каждый из которых соответствует примитивному типу данных)?