OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

10 – Operators and Decision Constructs (Часть 2)

Дмитрий Коган

Как меня слышно и видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели :

- Займёмся арифметическими операциями
- Вспомним особенности присваивания
- Углубимся в кастинг

Начинаем?

Темы экзамена

- ❑ Java Basics
- ❑ Working with Java Data Types
- ❑ Using Operators and Decision Constructs
- ❑ Creating and Using Arrays
- ❑ Using Loop Constructs
- ❑ Working with Methods and Encapsulation
- ❑ Working with Inheritance
- ❑ Handling Exceptions
- ❑ Working with Selected classes from the Java API

Подтемы экзамена

Using Operators and Decision Constructs

- **Use Java operators; use parentheses to override operator precedence**
- **Test equality between Strings and other objects using == and equals ()**
- **Create if and if/else and ternary constructs**
- **Use a switch statement**

Арифметические операторы

Binary numeric promotion

Binary numeric promotion implicitly applies appropriate widening primitive conversions so that a pair of operands have the widest numeric type of the two, which is always at least `int`. If `T` is the widest numeric type of the two operands after any unboxing conversions have been performed, the operands are promoted as follows during binary numeric promotion:

> If `T` is wider than `int`, both operands are converted to `T`; otherwise, both operands are converted to `int`.

This means that the resulting type of the operands is at least `int`.

Binary numeric promotion

Binary numeric promotion is applied in the following expressions:

- Operands of the arithmetic operators `*`, `/`, `%`, `+`, and `-`
- Operands of the relational operators `<`, `<=`, `>`, and `>=`
- Operands of the numerical equality operators `==` and `!=`
- Operands of the conditional operator `?` `:`, under certain circumstances

Арифметические операторы

Operator	Description
+	Adds two numeric values
-	Subtracts two numeric values
*	Multiplies two numeric values
/	Divides one numeric value by another
%	Modulus operator returns the remainder after division of one numeric value by another

Порядок вычисления

In the expression `a + b * c`, the operand `a` will always be fully evaluated before the operand `b`, which will always be fully evaluated before the operand `c`. However, the multiplication operator `*` will be applied before the addition operator `+`, respecting the precedence rules. Note that `a`, `b`, and `c` are arbitrary arithmetic expressions that have been determined to be the operands of the operators.

Порядок вычисления

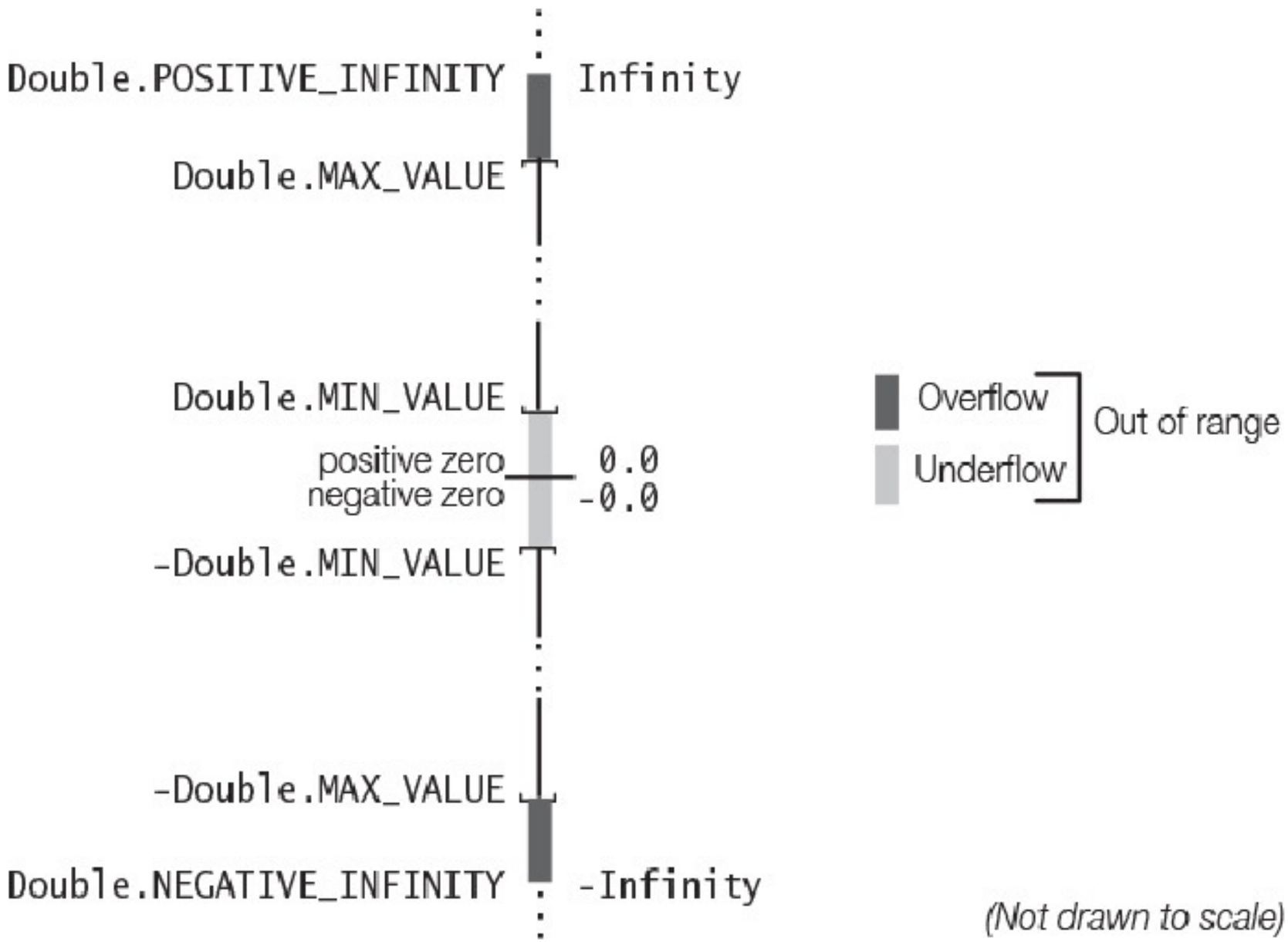
```
int fix = 10;
System.out.println(--fix + fix++); // Вычисляется слева направо
```

Границы значений

```
int tooBig   = Integer.MAX_VALUE + 1;     // -2147483648 which is
Integer.MIN_VALUE.
int tooSmall = Integer.MIN_VALUE - 1;     //  2147483647 which is
Integer.MAX_VALUE.
```

```
long notTooBig   = Integer.MAX_VALUE + 1L;   //  2147483648L in range.
long notTooSmall = Integer.MIN_VALUE - 1L;   // -2147483649L in range.
```

Границы значений



Границы значений

```
System.out.println( 4.0 / 0.0);          // Prints:  Infinity
System.out.println(-4.0 / 0.0);          // Prints: -Infinity
```

Negative zero compares equal to positive zero; in other words, (`-0.0 == 0.0`) is `true`.

Not a Number

```
System.out.println(0.0 / 0.0);          // Prints: NaN
```

NaN is represented by the constant named `NaN` in the wrapper classes `java.lang.Float` and `java.lang.Double`. Any operation involving NaN produces NaN. Any comparison (except inequality `!=`) involving NaN and any other value (including NaN) returns `false`. An inequality comparison of NaN with another value (including NaN) always returns `true`. However, the recommended way of checking a value for NaN is to use the static method `isNaN()` defined in both wrapper classes, `java.lang.Float` and `java.lang.Double`.

На дом

```
class MemorySaver {
    public static void main(String[] args) {
        int a = 123;
        int b = 321;
        System.out.println("a=" + a + ", b=" + b);              // a=123, b=321
        a = a + b;
        b = a - b;
        a = a - b;
        System.out.println("a=" + a + ", b=" + b);              // a=321, b=123
    }
}
```

Suppose, we declare

```
int a = Integer.MAX_VALUE;
int b = Integer.MIN_VALUE;
```

What will happen?

A. The algorithm will continue to work correctly
B. Both a and b will flip their signs
C. Compilation will fail
D. The code will throw an ArithmeticException

Умножаем

```
int    sameSigns     = -4    * -8;   // result:  32
double oppositeSigns =  4    * -8.0; // Widening of int 4 to double. result:
-32.0
int    zero          =  0    * -0;   // result:   0
```

Делим

```
int    i1 = 4  / 5;   // result: 0
int    i2 = 8  / 8;   // result: 1
double d1 = 12 / 8;   // result: 1.0; integer division, then widening
conversion
```

Оператор %

```
a % b = a - ( a / b ) * b
```

Поэтому,	5 % 3 = 2;	5 = 3*1 + 2	→ OK, это остаток
	5 % (-3) = 2;	5 ≠ -3*1 + 2	→ не остаток!
	(-5) % (3) = -2;	-5 ≠ 3*1 + -2	→ не остаток!
	(-5) % (-3) = -2;	-5 ≠ -3*1 + -2	→ остаток

Итак, остаток мы получим, когда оба операнда одинакового знака.

Важнее всего насчет %: Знак 1-го операнда и будет знаком результата.

Оператор %

```
int  r0 =  7  %  7;      //  0
int  r1 =  7  %  5;      //  2
long r2 =  7L % -5L;     //  2L
int  r3 = -7  %  5;      // -2
long r4 = -7L % -5L;     // -2L
boolean relation = -7L == (-7L / -5L) * -5L + r4;  // true
```

An `ArithmeticException` is thrown if the divisor evaluates to zero.

Оператор %

Note that the remainder operator accepts not only integral operands, but also floating-point operands. The *floating-point remainder* `r` is defined by the relation

r == *a* - (*b* * *q*)

where `a` and `b` are the dividend and the divisor, respectively, and `q` is the *integer* quotient of `(a/b)`. The following examples illustrate a floating-point remainder operation:

```
double  dr0 =  7.0  %  7.0;    //  0.0
float   fr1 =  7.0F %  5.0F;   //  2.0F
double  dr1 =  7.0  % -5.0;    //  2.0
float   fr2 = -7.0F %  5.0F;   // -2.0F
double  dr2 = -7.0  % -5.0;    // -2.0
boolean fpRelation = dr2  == (-7.0) - (-5.0) * (long)(-7.0 / -5.0);  // true
float   fr3 = -7.0F %  0.0F;   // NaN
```

+ и -

Arithmetic expression	Evaluation	Result when printed
3 + 2 - 1	((3 + 2) - 1)	4
2 + 6 * 7	(2 + (6 * 7))	44
-5 + 7 - -6	(((-5) + 7) - (-6))	8
2 + 4 / 5	(2 + (4 / 5))	2
13 % 5	(13 % 5)	3
11.5 % 2.5	(11.5 % 2.5)	1.5
10 / 0		ArithmeticException
2 + 4.0 / 5	(2.0 + (4.0 / 5.0))	2.8
4.0 / 0.0	(4.0 / 0.0)	Infinity
-4.0 / 0.0	((-4.0) / 0.0)	-Infinity
0.0 / 0.0	(0.0 / 0.0)	NaN

char

```
char char1 = 'a';
System.out.println(char1);
System.out.print(char1 + char1);
```

Outputs a

Outputs 194

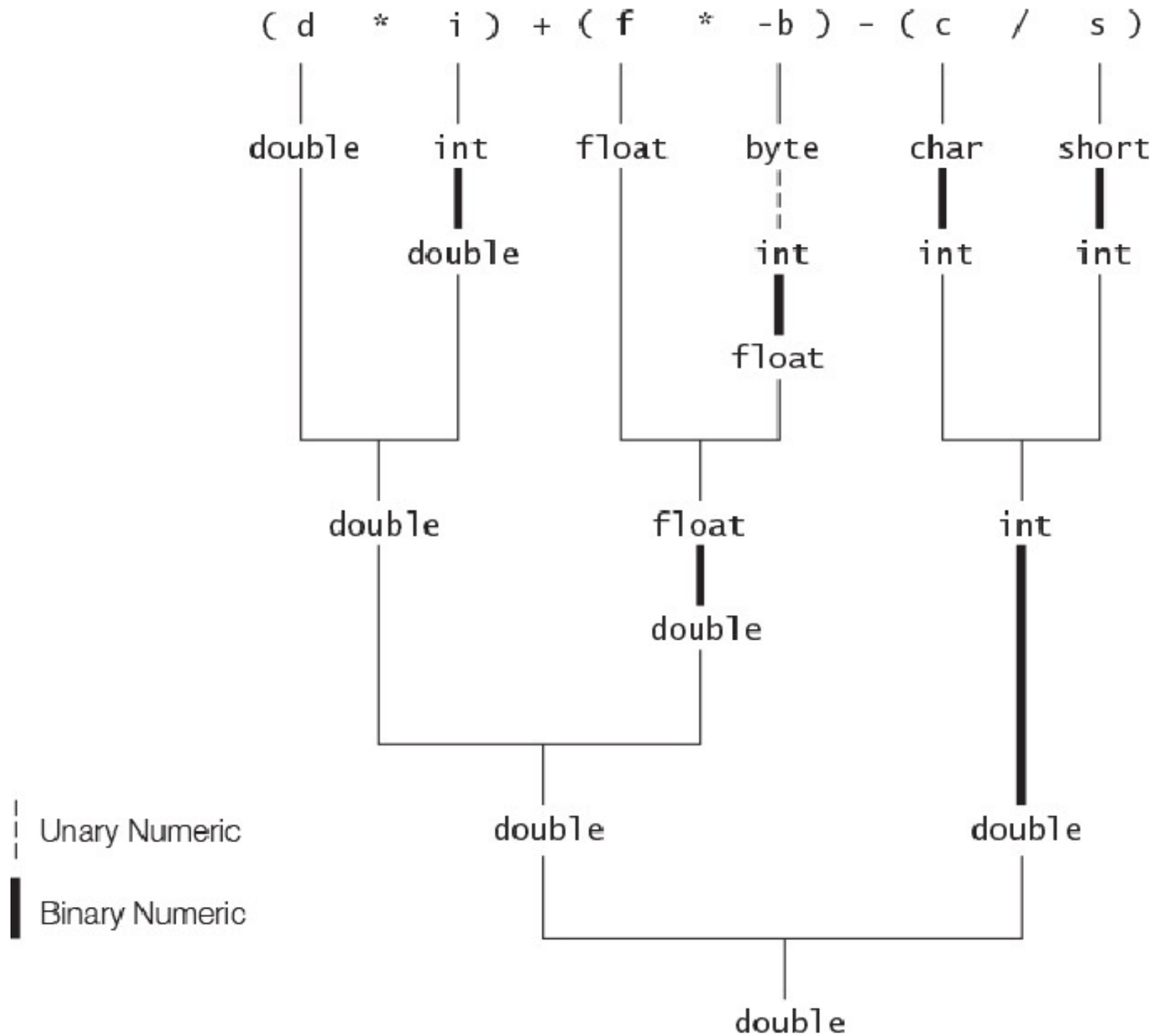
```
char char1 = 'a';
System.out.print(char1 - char1);
```

Outputs 0

Возвышение

```
public class NumPromotion {
  public static void main(String[] args) {
    byte   b = 32;
    char   c = 'z';                    // Unicode value 122 (\u007a)
    short  s = 256;
    int    i = 10000;
    float  f = 3.5F;
    double d = 0.5;
    double v = (d * i) + (f * -b) - (c / s);      // (1) 4888.0D
    System.out.println("Value of v: " + v);
  }
}
```

Возвышение


(d * i) + (f * -b) - (c / s)
double
int
float
byte
char
short
double
int
int
int
float
double
double
float
int
double
double
double
Unary Numeric
Binary Numeric

Возвышение

```
Byte   b = 10;        // Constant in range: narrowing and boxing on
assignment.
Short  s = 20;        // Constant in range: narrowing and boxing on
assignment.
char   c = 'z';       // 122 (\u007a)
int    i = s * b;     // Values in s and b promoted to int: unboxing,
widening.
long   n = 20L + s;   // Value in s promoted to long: unboxing, widening.
float  r = s + c;     // Value in s is unboxed. This short value and the char
                      // value in c are promoted to int, followed by implicit
                      // widening conversion of int to float on assignment.
double d = r + i;     // Value in i promoted to float, followed by implicit
                      // widening conversion of float to double on assignment.
```

Приведение типов

```
short h = 40;              // OK: int converted to short. Implicit narrowing.
h = h + 2;                 // Error: cannot assign an int to short.

h = (short) (h + 2);       // OK
```

```
h = (short) h + (short) 2;     // The resulting value should be cast.

h = (short) h + 2;             // The resulting value should be cast.
```

Упражнение

Given:

```
int myChar = 97;
int yourChar = 98;
System.out.print((char)myChar + (char)yourChar);

int age = 20;
System.out.print(" ");
System.out.print((float)age);
```

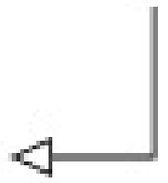
What is the output?

1. 195 20.0
2. 195 20
3. ab 20.0
4. ab 20
5. Compilation error

Ответ: 1

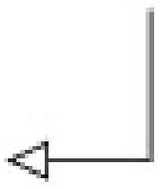
Константы

```
byte age1 = 10;
byte age2 = 20;
short sum = age1 + age2;
```



Fails to compile

```
final byte age1 = 10;
final byte age2 = 20;
short sum = age1 + age2;
```



Compiles successfully

Константы

```
final byte fb1 = 10;
// final fb1 = 30; // не пройдёт проверку компилятора, потому что 130 не влезает в byte
final byte fb2 = 100;
byte bsum = fb1 + fb2;
```

Вопросы?

Операторы присваивания

Простой оператор

Operator	Description
=	Assigns the value on the right to the variable on the left

Присваивание

- Операторы присваивания (assignment operators, assops, ассопы) записывают то или иное значение в переменную. Левый операнд должен быть соответствующей локальной переменной, элементом массива или полем. Справа же может стоять значение любого типа, совместимого с переменной.
- В отличие от всех прочих бинарных операторов, ассопы вычисляются справа налево, означая на практике, что цепочка присваиваний `a=b=c` вычисляется следующим образом: `a=(b=c)`.

Множественное присваивание

```
int j, k;
j = 10;          // j gets the value 10, which is returned
k = j;           // k gets the value of j, which is 10, and this value is
returned
k = j = 10;      // (k = (j = 10))
```

```
Pizza pizzaOne, pizzaTwo;
pizzaOne = pizzaTwo = new Pizza(“Supreme”); // Aliases
```

```
int v1 = v2 = 2016;              // Only v1 is declared. Compile-time error!
```

Кастинг

```
short tail = (short)(4 + 10);
long feathers = 10(long);  // DOES NOT COMPILE
```

```
float egg = 2.0 / 9;          // DOES NOT COMPILE
int tadpole = (int)5 * 2L;    // DOES NOT COMPILE
short frog = 3 - 2.0;         // DOES NOT COMPILE
```

Кастинг

```
int fish = 1.0;          // DOES NOT COMPILE
short bird = 1921222;    // DOES NOT COMPILE
int mammal = 9f;         // DOES NOT COMPILE
long reptile = 192301398193810323;  // DOES NOT COMPILE
```

```
int trainer = (int)1.0;
short ticketTaker = (short)1921222;  // Stored as 20678
int usher = (int)9f;
long manager = 192301398193810323L;
```

Упражнение

What is the output of the following program?

```
1: public class CandyCounter {
2:    static long addCandy(double fruit, float vegetables) {
3:       return (int)fruit+vegetables;
4:    }
5:
6:    public static void main(String[] args) {
7:       System.out.print(addCandy(1.4, 2.4f) + "-");
8:       System.out.print(addCandy(1.9, (float)4) + "-");
9:       System.out.print(addCandy((long)(int)(short)2, (float)4)); } }
```

A. `4-6-6.0`

B. `3-5-6`

C. `3-6-6`

D. `4-5-6`

E. The code does not compile because of line 9.

F. None of the above

Ответ: F

Упаковка

```
Boolean   boolRef = true;  // Boxing.
Byte      bRef =  2;       // Constant in range: narrowing, then boxing.
// Byte  bRef2 =  257;     // Constant not in range. Compile-time error!

short s = 10;              // Narrowing from int to short.
// Integer   iRef1 = s;    // short not assignable to Integer.
Integer iRef3 = (int) s;   // Explicit widening with cast to int and boxing

boolean bv1 = boolRef;     // Unboxing.
byte  b1 = bRef;           // Unboxing.
int   iVal = bRef;         // Unboxing and widening.

Integer iRefVal = null;            // Always allowed.
// int j = iRefVal;                // NullPointerException at runtime.
if (iRef3 != null) iVal = iRef3;  // Avoid exception at runtime.
```

Составные операторы

Operator	Description
+=	Adds the value on the right to the variable on the left and assigns the sum to the variable
-=	Subtracts the value on the right from the variable on the left and assigns the difference to the variable
*=	Multiplies the value on the right with the variable on the left and assigns the product to the variable
/=	Divides the variable on the left by the value on the right and assigns the quotient to the variable

Составные операторы

- Составные (compound) ассопы, напр., `a+=`*`smth`* и т.п., эквивалентны операции `a=a+`*`smth`* и т.д. Иными словами, += можно трактовать так: «сначала сложи, затем присвой».

Составные операторы

$$variable\ op{=}\ expression$$

$$variable = \quad (type) \quad ((variable)\ op\ (expression))$$

Составные операторы

Expression	Given T as the numeric type of x, the expression is evaluated as:
x *= a	x = (T) ((x) * (a))
x /= a	x = (T) ((x) / (a))
x %= a	x = (T) ((x) % (a))
x += a	x = (T) ((x) + (a))
x -= a	x = (T) ((x) - (a))

Составные операторы

```
int i = 2;
i *= i + 4;                    // (1) Evaluated as i = (int) ((i) * (i + 4)).

Integer iRef = 2;
iRef *= iRef + 4;              // (2) Evaluated as iRef = (Integer) ((iRef) * (iRef
+ 4)).

byte b = 2;
b += 10;                       // (3) Evaluated as b = (byte) (b + 10).
b = b + 10;                    // (4) Will not compile. Cast is required.
```

Составные операторы

```
int[] a = new int[] { 2015, 2016, 2017 };
int i = 2;
a[i] += 1;       // Evaluates as a[2] = a[2] + 1, and a[2] gets the value
                    2018.
```

Упражнение

Given the following code snippet, what is the value of the variables after it is executed? (Choose all that apply.)

```
int ticketsTaken = 1;
int ticketsSold = 3;
ticketsSold += 1 + ticketsTaken++;
ticketsTaken *= 2;
ticketsSold += (long)1;
```

A. `ticketsSold` is 8

B. `ticketsTaken` is 2

C. `ticketsSold` is 6

D. `ticketsTaken` is 6

E. `ticketsSold` is 7

F. `ticketsTaken` is 4

G. The code does not compile.

Ответ: CF

Добавим инкремент

```
int test = 1;
test += test++ - ++test;
System.out.println(test);
```

Возвращаемое значение

```
class Boo {
    public static void main(String[] args) {
        int a, b = 0;
        boolean boo;
        a = (boo = true) ? b = 10 : 11;
        System.out.println(a + " " + boo + " " + b);
    }
}
```

Возвращаемое значение

```
class Boo {
    public static void main(String[] args) {
        int a, b = 0;
        boolean boo;
        a = (boo = true) ? b = 10 : 11;
        System.out.println(a + " " + boo + " " + b);      // 10 true 10
    }
}
```

Любой ассоп можно рассматривать так: когда какой-то переменной присваивается значение, то процесс на этой переменной не заканчивается; значение словно «просачивается» еще дальше, левее. А тот факт, что переменная (вроде **boo** или **b** в приведенном приме) получает по ходу дела некое значение, есть лишь побочный результат. Становится ясно, что – по крайней мере, с точки зрения тернопа – выражение **boo** = `true` эквивалентно значению `true`, и выражение **b** = 10 тоже производит значение, которое можно присвоить переменной **a**.

Вопросы?

Домашнее задание

Тест

Пожалуйста, пройдите опрос

https://otus.ru/polls/17816/

Спасибо за внимание!

Складывайте и приумножайте!