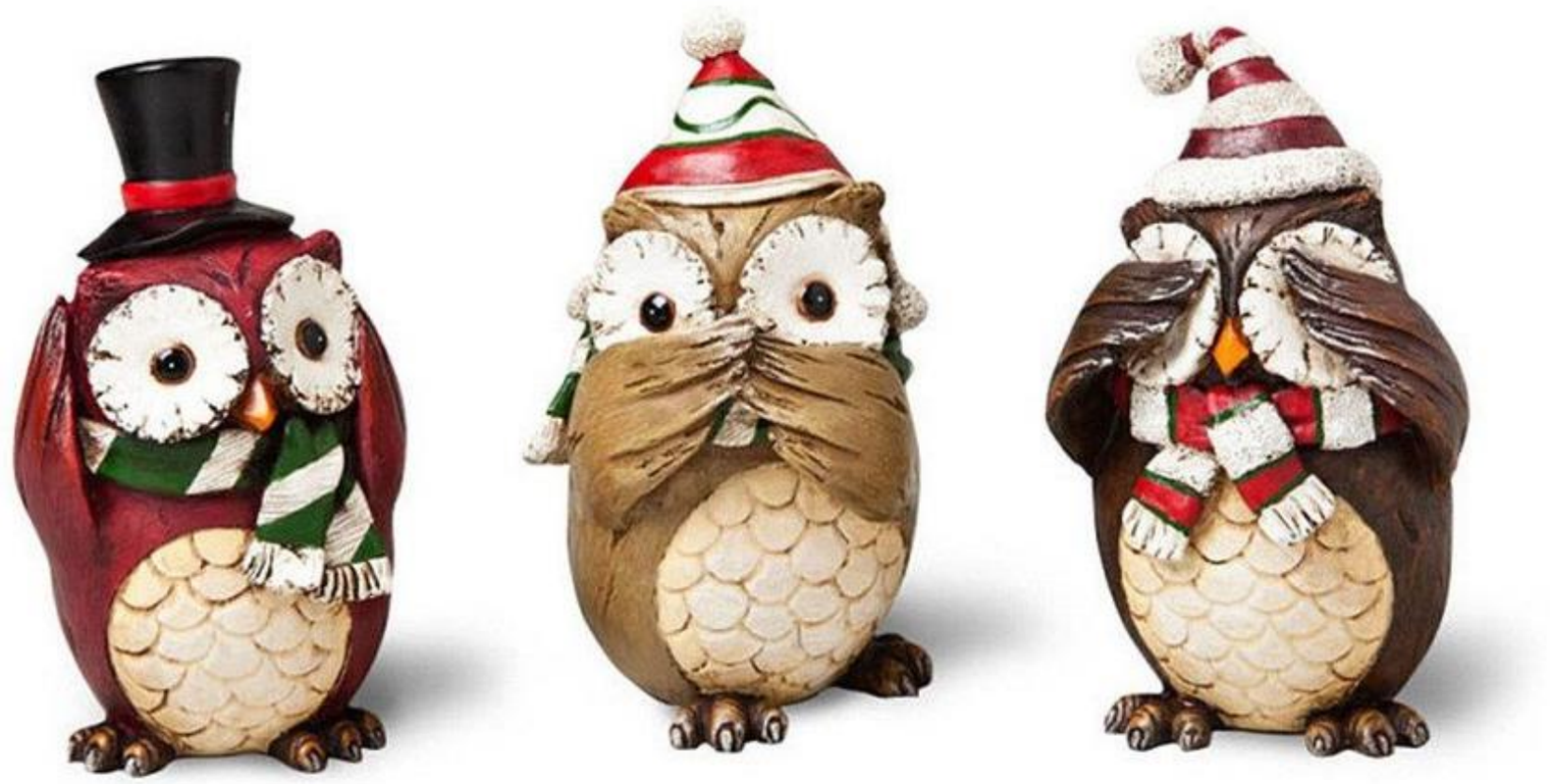
OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

11 – Operators and Decision Constructs (Часть 3)

Дмитрий Коган

Как меня слышно и видно?



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

Цели :

- Раскроем секреты конкатенации
- Отличим равное от неравного
- Заглянем в генеалогию объектов

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**

Конкатенация

Перегруженный +

- На экзамене есть вопрос, который интересуется – среди всего прочего, – допускает ли Джава перегрузку операторов. Правильный ответ «Нет, не допускает». Что любопытно, в Джаве все-таки есть нечто, напоминающее перегрузку оператора +: он как бы перегружен под **String**.

Когда нет вопросов

```
String animal = "Gray " + "elephant";
```

Правило конкатенации

If either operand is a `String`*, the* `+` *operator becomes a* `String` *concatenation operator. If both operands are numbers, the* `+` *operator is the addition operator.*

Применяем правило

```
String a = "String";
int b = 3;
int c = 7;
System.out.println(a + b + c);
```

```
System.out.println(a + (b + c));
```

Применяем правило

```
String a = "String";
int b = 3;
int c = 7;
System.out.println(a + b + c);
```

String37

```
System.out.println(a + (b + c));
```

String10

Применяем правило

```
String strVal = "" + 2016;
String theName = " Uranium";
theName = " Pure" + theName;
String trademark1 = 100 + "%" + theName;
```

```
String trademark2 = 100 + '%' + theName;

System.out.println("2 * 2 = " + 2 * 2);
```

Применяем правило

```
String strVal = "" + 2016;                          // (1) "2016"
String theName = " Uranium";
theName = " Pure" + theName;                        // (2) " Pure Uranium"
String trademark1 = 100 + "%" + theName;            // (3) "100% Pure Uranium"

String trademark2 = 100 + '%' + theName;            // (4) "137 Pure Uranium"

System.out.println("2 * 2 = " + 2 * 2);             // (7) 2 * 2 = 4
```

Неясная ситуация

```
System.out.println(x.foo() + 7);
```

Составной оператор

```
String s = "123";
s += "45";
s += 67;
System.out.println(s);
```

Составной оператор

```
String s = "123";
s += "45";
s += 67;
System.out.println(s);
```

```
1234567
```

Перевод в строку

For an operand of a primitive data type, its value is converted to a string representation.

For all reference value operands, a string representation is constructed by calling the no-argument `toString()` method on the referred object. Most classes override this method from the `Object` class so as to provide a more meaningful string representation of their objects.

true, false, null

Values like `true`, `false`, and `null` have string representations that correspond to their names. A reference variable with the value `null` also has the string representation `"null"` in this context.

null +

- `null` – единственный литерал, который может быть присвоен любому ссылочному типу, например, **String**-переменной и т.п. С другой стороны, `null` нельзя присваивать примитиву.
- Если **String**-объект равен `null` и затем этот объект будет использован с оператором +, результатом окажется "null*плюс-что-то-еще*".

```
String concat = null + "-a-" + true;
System.out.println(concat); // null-a-true
```

Скомпилируется?

```
String str = "";
str = null + 'a';
str += 'a';
str += null + 'a';
str = str + null + 'a';
```

Ошибки компиляции

```
String str = "";
str = null + 'a';                    // INVALID
str += 'a';
str += null + 'a';                   // INVALID
str = str + null + 'a';
```

Определение типов

```
Object sobj = "";
String str = "";
str = sobj + 'a';
str = (String) sobj + 'a';
```

Определение типов

```
Object sobj = "";
String str = "";
str = sobj + 'a';                    // INVALID
str = (String) sobj + 'a';           // прекрасно компилируется
```

Мнемоника

- **String** с плюсом (включая +=) берет что угодно в любом порядке; напр., `str += 'a'` или `str += false` или `str +=1` . Все это компилируется, как и `str = 'a' + str; str = 'b' + 63 + "a"; str = null + str;` и т.д.

- Если выражение содержит **String**, за которым идет +, всё, что стоит справа от этого стринга, автоматически превратится в цепочку стрингов. Впрочем, у круглых скобок может быть свое мнение на этот счет.

Упражнение

```
class Feline {
  public static void main(String[] args) {
    long x = 42L;
    long y = 44L;
    System.out.print(" " + 7 + 2 + " ");
    System.out.print(foo() + x + 5 + " ");
    System.out.println(x + y + foo());
  }
  static String foo() { return "foo"; }
}
```

What is the result?

A. `9 foo47 86foo`

B. `9 foo47 4244foo`

C. `9 foo425 86foo`

D. `9 foo425 4244foo`

E. `72 foo47 86foo`

F. `72 foo47 4244foo`

G. `72 foo425 86foo`

H. `72 foo425 4244foo`

I. Compilation fails

Ответ: G

Вопросы?

Операторы (не-)равенства

Равно или не равно

Operator	Apply to primitives	Apply to objects
`==`	Returns `true` if the two values represent the same value	Returns `true` if the two values reference the same object
`!=`	Returns `true` if the two values represent different values	Returns `true` if the two values do not reference the same object

Три сценария

- Comparing two numeric or character primitive types. If the numeric values are of different data types, the values are automatically promoted. For example, `5 == 5.00` returns `true` since the left side is promoted to a `double`.
- Comparing two `boolean` values
- Comparing two objects, including `null` and `String` values

Примитивы

- Попытка сравнить несовместимые типы всегда вызывает комперр. К примеру, `int`, `char` или значение с плавающей точкой нельзя сравнивать с `boolean`. С другой стороны, мы можем сравнивать `int` с `char`'ом, поскольку `char` – это целочисленный тип и ведет себя практически как `short` без знака.
- Операторы 'равно' (==) и 'не равно' (!=) можно применять к любой паре совместимых примитивов.
- == возвращает `true`, если сравниваемые примитивы равны между собой.
- != возвращает `true`, если сравниваемые примитивы не равны между собой.

Совместимые типы

```
class ComparePrimitives {
  public static void main(String[] args) {
    System.out.println("char 'a' == 'a'? " + ('a' == 'a'));
    System.out.println("char 'a' == 'b'? " + ('a' == 'b'));
    System.out.println("5 != 6? " + (5 != 6));
    System.out.println("5.0 == 5L? " + (5.0 == 5L));
    System.out.println("true == false? " + (true == false));
  }
}
```

```
char 'a' == 'a'? true
char 'a' == 'b'? false
5 != 6? true
5.0 == 5L? true
true == false? false
```

Несовместимые типы

```
boolean monkey = true == 3;          // DOES NOT COMPILE
boolean ape = false != "Grape";      // DOES NOT COMPILE
boolean gorilla = 10.2 == "Koko"; // DOES NOT COMPILE
```

Присвоить на лету

```
boolean bear = false;
boolean polar = (bear = true);
System.out.println(polar);  // true
```

Примеры

```
int year = 2002;
boolean isEven  = year % 2 == 0;

boolean compare = '1' == 1;

boolean test    = compare == false;
```

Примеры

```
int year = 2002;
boolean isEven  = year % 2 == 0;      // true.
boolean compare = '1' == 1;           // false. Binary numeric promotion
applied.
boolean test    = compare == false;   // true.
```

Погрешность

Care must be exercised when comparing floating-point numbers for equality, as an infinite number of floating-point values can be stored only as approximations in a finite number of bits. For example, the expression `(1.0 - 2.0/3.0 == 1.0/3.0)` returns `false`, although mathematically the result should be true.

Ассоциативность

Analogous to the discussion for relational operators, mathematical expressions like *a* = *b* = *c* must be written using relational and logical/conditional operators. Since equality operators have left associativity, the evaluation of the expression `a == b == c` would proceed as follows: `((a == b) == c)`. Evaluation of `(a == b)` would yield a `boolean` value that *is* permitted as an operand of a data value equality operator, but (*boolean value* == `c`) would be illegal if `c` had a numeric type. This problem is illustrated in the following examples. The expression at (1) is illegal, but those at (2) and (3) are legal.

```
int a, b, c;
a = b = c = 5;
boolean illegal = a == b == c;                  // (1) Illegal.
boolean valid2 = a == b && b == c;              // (2) Legal.
boolean valid3 = a == b == true;                // (3) Legal.
```

Объекты

```
File x = new File("myFile.txt");
File y = new File("myFile.txt");
File z = x;
System.out.println(x == y);  // Outputs false
System.out.println(x == z);  // Outputs true
```

null

```
if (objRef != null) {
    // … use objRef …
}
```

```
System.out.print(null == null);  // true
```

Ссылки или примитивы

Note that only when the type of *both* operands is either a reference type or the `null` type, do these operators test for object reference equality. Otherwise, they test for primitive data equality.

```
Integer iRef = 10;
boolean b1 = iRef == null;          // (1) Object reference equality
boolean b2 = iRef == 10;            // (2) Primitive data equality
boolean b3 = null == 10;            // Compile-time error!
```

Вопросы?

Операторы сравнения

Реляционные операторы

Operator	Description
`<`	Returns `true` if the value on the left is strictly less than the value on the right
`<=`	Returns `true` if the value on the left is less than or equal to the value on the right
`>`	Returns `true` if the value on the left is strictly greater than the value on the right
`>=`	Returns `true` if the value on the left is greater than or equal to the value on the right
a `instanceof` *b*	Returns `true` if the reference that `a` points to is an instance of a class, subclass, or class that implements a particular interface, as named in `b`

Сравниваем числа

```
int gibbonNumFeet = 2, wolfNumFeet = 4, ostrichNumFeet = 2;
System.out.println(gibbonNumFeet < wolfNumFeet);      // true
System.out.println(gibbonNumFeet <= wolfNumFeet);     // true
System.out.println(gibbonNumFeet >= ostrichNumFeet);  // true
System.out.println(gibbonNumFeet > ostrichNumFeet);   // false
```

char

```
boolean b1 = 'A' < 'B'; // Character literals
boolean b2 = '\u0041' < '\u0042'; // Unicode literals
boolean b3 = 0x0041 < 0x0042; // Hexadecimal literals
boolean b4 = 65 < 66; // Integer literals that fit in a char
boolean b5 = 0101 < 0102; //Octal literals
boolean b6 = '\101' < '\102'; // Octal literals
boolean b7 = 'A' < 0102; // Character and Octal literals
```

Числа с плавающей точкой

```
boolean b1 = 9.00D < 9.50D; // Floating points with D postfixes
boolean b2 = 9.00d < 9.50d; // Floating points with d postfixes
boolean b3 = 9.00F < 9.50F; // Floating points with F postfixes
boolean b4 = 9.0f < 9.50f; // Floating points with f postfixes
boolean b5 = (double)9 < (double)10; // Integers with explicit casts
boolean b6 = (float)9 < (float)10; // Integers with explicit casts
boolean b7 = 9 < 10; // Integers that fit into floating points
boolean b8 = (9d < 10f);
boolean b9 = (float)11 < 12;
```

Упаковка

```
double hours = 45.5;
Double time = 18.0;                 // Boxing of double value.
boolean overtime = hours >= 35; // true. Binary numeric promotion: double <-
int.
boolean beforeMidnight = time < 24.0;// true. Unboxing of value in time
reference.
char letterA = 'A';
boolean order = letterA < 'a';  // true. Binary numeric promotion: int <-
char.
```

Ассоциативность

Relational operators are nonassociative. Mathematical expressions like $a \leq b \leq c$ must be written using relational and boolean logical/conditional operators.

```
int a = 1, b = 7, c = 10;
boolean illegal = a <= b <= c;        // (1) Illegal.
boolean valid2 = a <= b && b <= c;    // (2) OK.
```

Since relational operators have left associativity, the evaluation of the expression `a <= b <= c` at (1) in these examples would proceed as follows: `((a <= b) <= c)`. Evaluation of `(a <= b)` would yield a `boolean` value that is not permitted as an operand of a relational operator; that is, `(`*boolean value* `<= c)` would be illegal.

Упражнение

```
System.out.println( 10 + 5 == 4 + 11);
System.out.println( 10 + (5 == 4) + 11);
System.out.println( "" + (5 <= 4) + "" );
System.out.println( "" + 10 + 5 == 4 + 11 + "");
```

How many statements fail compilation?

A. One
B. Two
C. Three
D. None

Ответ: А

Последняя строка

```
System.out.println("" + 10 + 5 == 4 + 11 + "");
System.out.println("" + 10 + 5 == 55 + 50 + "");
```

Последняя строка

```
System.out.println("" + 10 + 5 == 4 + 11 + "");  // Всегда false: слева - строка "105", справа - строка "15"
System.out.println("" + 10 + 5 == 55 + 50 + ""); // Всегда true: слева и справа - строка "105"
```

instanceof

```
Integer zooTime = Integer.valueOf(9);
Number num = zooTime;
Object obj = zooTime;
```

instanceof

```
public static void openZoo(Number time) {
    if(time instanceof Integer)
        System.out.print((Integer)time + " O'clock");
    else
        System.out.print(time);
}
```

Ошибка компиляции

```
public static void openZoo(Number time) {
if(time instanceof String) // DOES NOT COMPILE
...
```

null

```
System.out.print(null instanceof Object);

Object noObjectHere = null;
System.out.print(noObjectHere instanceof String);
```

The preceding examples both print `false`. It almost doesn't matter what the right side of the expression is. We say "almost" because there are exceptions. The last example does not compile, since `null` is used on the right side of the `instanceof` operator:

```
System.out.print(null instanceof null);  // DOES NOT COMPILE
```

Таблица

First Operand (Reference Being Tested)	instanceof Operand (Type We're Comparing the Reference Against)	Result
`null`	Any class or interface type	`false`
`Foo` instance	`Foo, Bar, Face, Object`	`true`
`Bar` instance	`Bar, Face, Object`	`true`
`Bar` instance	`Foo`	`false`
`Foo [ ]`	`Foo, Bar, Face`	`compiler error`
`Foo [ ]`	`Object`	`true`
`Foo [ 1 ]`	`Foo, Bar, Face, Object`	`true`

```
interface Face { }
class Bar implements Face{ }
class Foo extends Bar { }
```

Вопросы?

Домашнее задание

Тест

Пожалуйста, пройдите опрос

https://otus.ru/polls/17817/

Спасибо за внимание!

Сравнивайте лучшее!