Java

Тема: Типы данных и операторы

Курс «Основы программирования на Java»

Автор: А. Е. Анисимов, УдГУ

Тема: Типы данных и операторы

A. Базовые типы данных. Переменные
B. Операции
C. Классы- оболочки
D. Операторы
E. Класс Math

A. Базовые типы данных. Переменные

Язык Java является объектно-ориентированным, но существуют типы данных (простые/примитивные), не являющиеся объектами.

- Фактор производительности

Простые типы делятся на 4 группы:

- целые: int, byte, short, long,
- числа с плавающей точкой: float, double
- символы: char
- логические: boolean

Базовые (необъектные, примитивные) типы данных

Примитив-ный тип	Размер* (бит)	Мин. значение	Макс. значение	Класс-оболочка
boolean	-	-	-	Boolean
char	16	Unicode 0	U2^16-1	Character
byte	8	-128	127	Byte
short	16	-32768	32767	Short
int	32	-2147483648	2147483647	Integer
long	64	-9223372036854775808	9223372036854775807	Long
float	32	-3.4E+38	3.4E+38	Float
double	64	-1.7E+308	1.7E+308	Double
void	-	-	-	Void

*Размер одинаков для всех платформ; за счет этого становится возможной переносимость кода

Литералы

Целые	Вещественные (с плав. точкой)	Символьный	Логический
byte short int long	float double	char	boolean
1, 2, 3, -2 012 0x23f 2553L	3.0F .9937F 3.455E8 1.0D	‘s’ ‘\141’ ‘\u0061’ ‘\n’	true false

Переменные:

- Основное место для хранения данных
- Должны быть явно объявлены
- Каждая переменная имеет тип, идентификатор и область видимости
- Определяются для класса, для экземпляра и внутри метода
- Имя может содержать буквы, цифры, знаки _ и $. С цифры начинаться не может.
- Основная форма объявления:

```
тип идентификатор [ = значение];
```

Значения по умолчанию

Примитивный тип	Значение по умолчанию*
boolean	false
char	'\u0000' (null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

*инициализируются автоматически

Пример программы 2.01.

```
/*Пример 2.01.
   Программа – описание переменных простых типов*/

public class VariablesExample {
      public static void main(String[] args) {
            int count = 10;
            float price = 11.0f;
            int i, j, k = 1;
            double radiusCircle;
      }
}
```

Преобразование типа:

- при смешивании операндов разных типов в выражениях

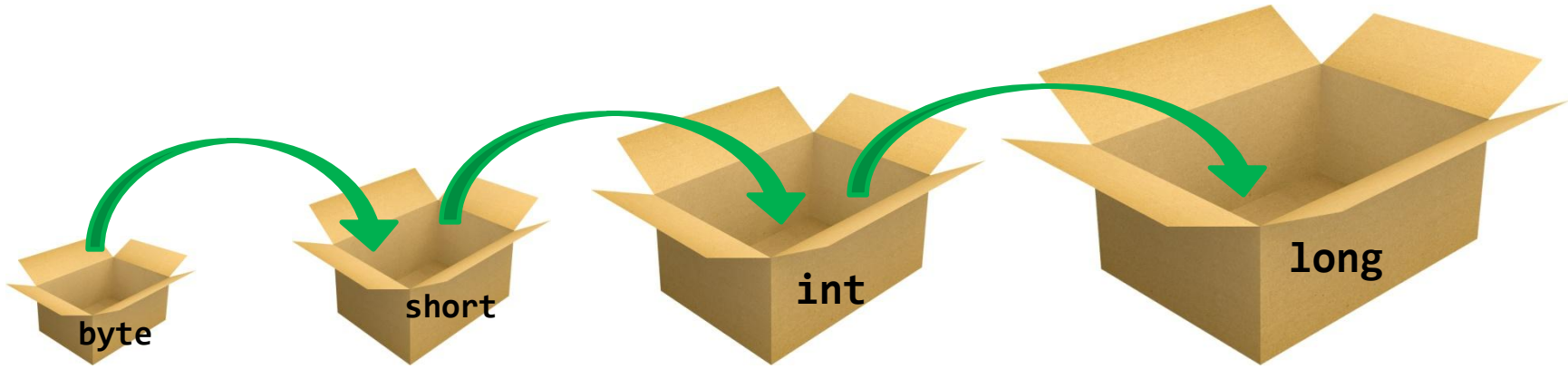
- преобразование бывает *явное* и *неявное* (автоматическое)

```
int x = 1;
byte y = (byte) x; //явное преобразование
int z = y;         //неявное преобразование
```

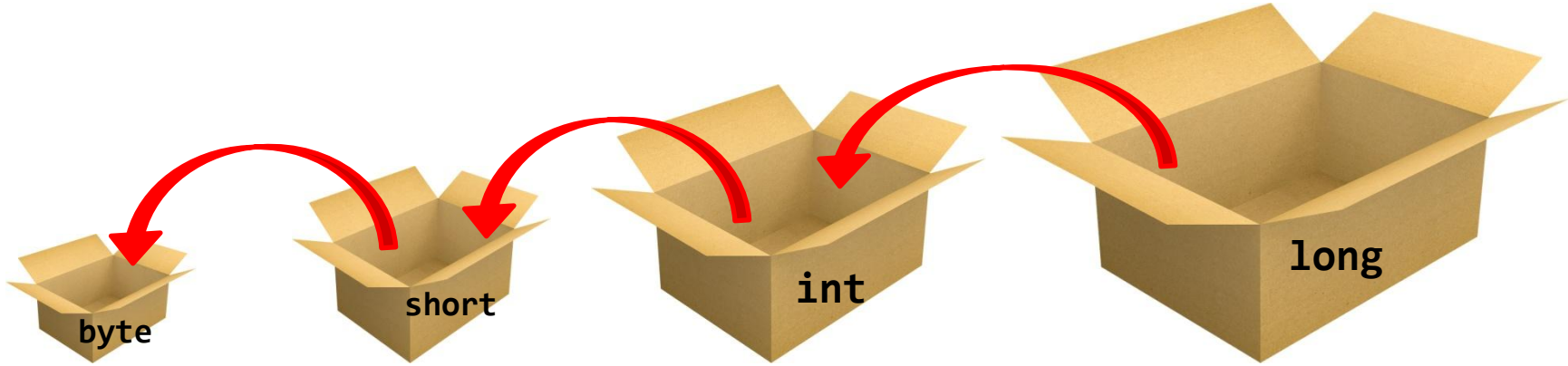
- преобразование бывает
  - *расширяющее* (от меньшего типа к большему)
  - *сужающее* (от большего типа к меньшему)

```
int x = 1;
byte y = (byte) x; //сужающее преобразование
int z = (int)y;    //расширяющее преобразование
```

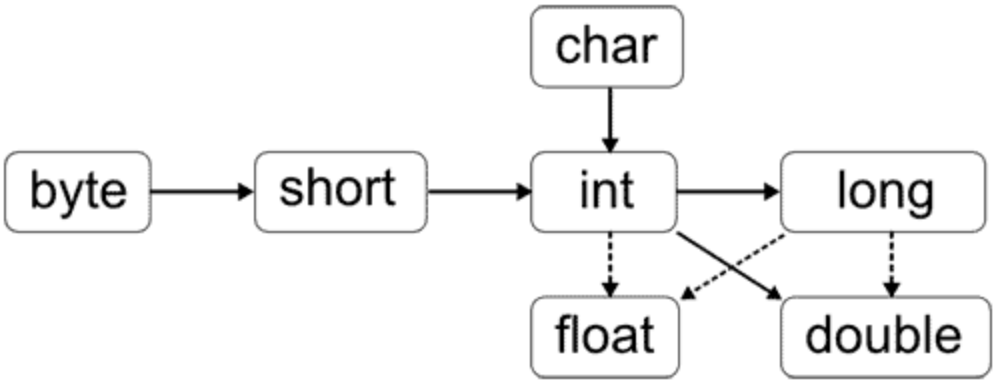
Расширяющее преобразование – неявно



Сужающее преобразование – только явно и возможно искажение!



- **Расширяющее (повышающее) преобразование** - производится неявно, автоматически, так как не происходит искажения значения. Правило для расширяющего преобразования:



(пунктир означает, что может произойти потеря точности)

- При **сужающем преобразовании** возможно искажение значения, поэтому оно всегда выполняется явно.
- При выполнении арифметических операций (кроме инкремента, декремента и присваивания с операцией) Java приводит выражение к типу по умолчанию (int, double)

Пример программы 2.02.

```
/*Пример 2.02. Приведение типов данных */
public class LoseAccuracy {
public static void main(String[] args) {
      byte b = 10;
      long l;
      l = b; //допустимо - повышающее преобразование

      b = b*2; //ошибка! правая часть приведена к int
      b = (byte) (b*2);
      b++;              //допустимо
      int i = b*b*b; //допустимо

      double pi = 3.14;
      i = (int)pi; // i == 3 - усечение

      float f = 1.234567890f;
      double d = f; // d == 1.2345678806304932
}
```

В множество значений типов с плавающей точкой (**`float, double`**) введены:

- Положительная бесконечность (**`+Infinity`**) – **`результат деления положительного числа на 0`**
- Отрицательная бесконечность (**`-Infinity`**)

  `результат деления отрицательного числа на 0`
- Не число (NaN)

  `извлечение квадратного корня из отрицательного числа или деление 0.0 на 0.0`

Пример программы 2.03.

```
/*Пример 2.03. Бесконечности и не числа */
public class InfinityExample{
public static void main(String[] args) {
    double d = 1.0;
    System.out.println("d/0=" + d / 0);
    System.out.println("-d/0=" + -d / 0);
    System.out.println("sqrt(-1)=" + Math.sqrt(-d));
}
```

Console

```
d/0=Infinity
-d/0=-Infinity
sqrt(-1)=NaN
```

B. Операции

Операции над примитивными типам в Java:

- присваивание
- арифметические
- битовые
- логические
- отношения
- и др.

Арифметические операции

+	Сложение	**/**	Деление
+=	Сложение (с присваиванием)	**/=**	Деление (с присваиванием)
–	Бинарное вычитание и унарное изменение знака	**%**	Деление по модулю
–=	Вычитание (с присваиванием)	**%=**	Деление по модулю (с присваиванием)
*****	Умножение	**++**	Инкремент
***=**	Умножение (с присваиванием)	**--**	Декремент

Битовые операции

|	Или	**>>**	Сдвиг вправо
|=	Или (с присваиванием)	**>>=**	Сдвиг вправо (с присваиванием)
&	И	**>>>**	Сдвиг вправо с появлением нулей
&=	И (с присваиванием)	**>>>=**	Сдвиг вправо с появлением нулей и присваиванием
^	Исключающее или	**<<**	Сдвиг влево
^=	Исключающее или (с присваиванием)	**<<=**	Сдвиг влево с присваиванием
~	Унарное отрицание		

Операции отношения

<	Меньше	>	Больше
<=	Меньше либо равно	>=	Больше либо равно
==	Равно	!=	Не равно

Логические операции

||	Или	&&	И
!	Унарное отрицание		

И др.

instanseof	Принадлежность классу	new	Создание объекта
[ ]	Индекс	()	Вызов метода
? :	Условная операция	.	Доступ

Таблица приоритетов операций

№	Операция	Порядок выполнения
1	**[ ] . () (вызов метода)**	Слева направо
2	**! ~ ++ -- +(унарный) -(унарный) () (приведение) new**	Справа налево
3	*** / %**	Слева направо
4	**+ -**	Слева направо
5	**<< >> >>>**	Слева направо
6	**< <= > >= instanceof**	Слева направо
7	**== !=**	Слева направо
8	**&**	Слева направо
9	**^**	Слева направо
10	**|**	Слева направо
11	**&&**	Слева направо
12	**||**	Слева направо
13	**?:**	Слева направо
14	**= += -= *= /= %= |= ^= <<= >>= >>>=**	Справа налево

Пример программы 2.04.

```
/*Пример 2.04. Операции  */
public class Operations {
public static void main(String[] args) {
    int a, b, c;
    a = 1234;
    b = a / 100 % 10; // a == 2

    a = 4;            // a == 100 (в 2 системе счисления)
    b = 2;            // b == 10  (в 2 системе счисления)
    c = a | b;        // c == 110 (в 2 системе счисления)

    c = a << b;       // c == 10000(в 2 системе счисл.)

    c = a / (b - 2); // генерация исключения
                      //java.lang.ArithmeticException
}
```

В языке Java существуют константы

```
Double.POSITIVE_INFINITY;
Float.POSITIVE_INFINITY;
Double.NEGATIVE_INFINITY;
Float.NEGATIVE_INFINITY;
Double.NaN;
Float.NaN;
```

которые можно использовать для проверки результатов вещественных операций деления на 0 или извлечения квадратного корня из отрицательного числа

Пример программы 2.05.

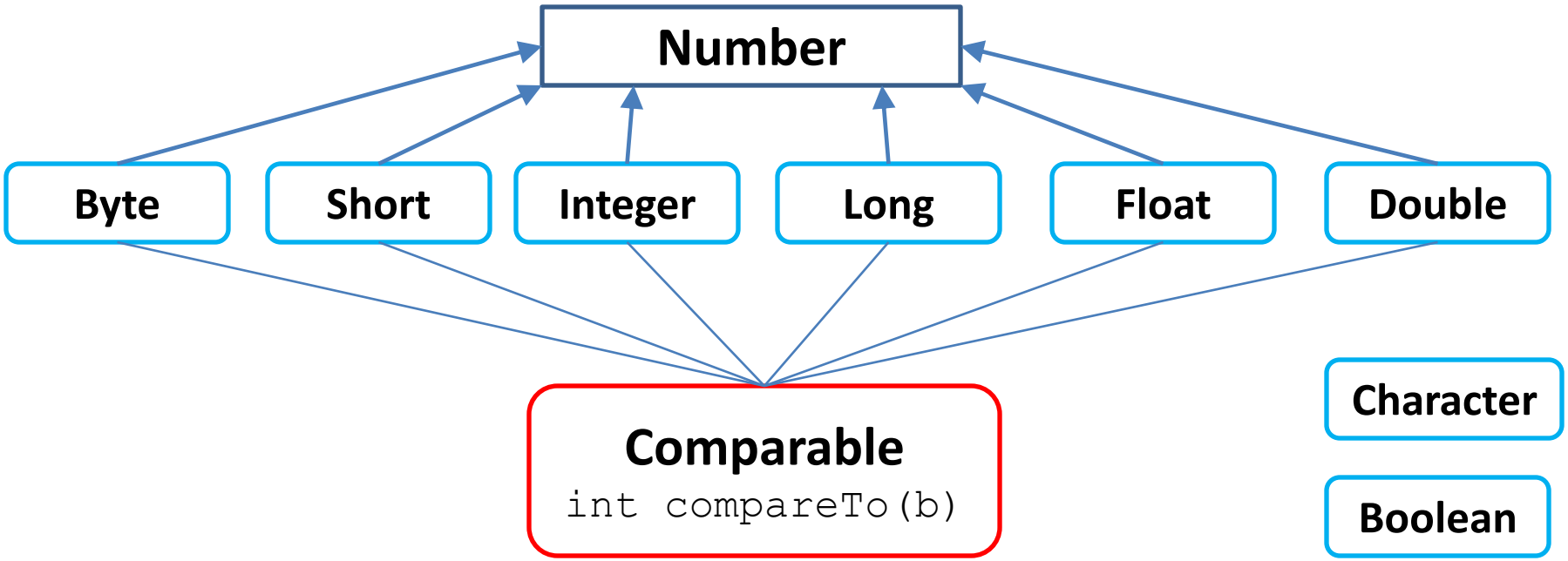
```
/*Пример 2.05. Неопределенность и бесконечность*/
public class InfinityUncertain{
public static void main(String[] args) {

    double d = 1.0 / 0;
    // d = - 1.0 / 0;
    // d = Math.sqrt(-1.0)

    if (d == Double.POSITIVE_INFINITY)
       System.out.println("Positive infinity");

    if (d == Double.NEGATIVE_INFINITY)
       System.out.println("Negative infinity");

    if (Double.isNaN(d))  //неверно так: d == Double.NaN)
       System.out.println("Not a number");
}
```

C. Классы - оболочки

Для каждого примитивного типа в Java имеется соответствующий ему класс, объекты которого могут хранить те же значения. Такие называются классы называются оболочками (обертками).

Примитивный тип	Класс-оболочка
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

- Числовые классы-оболочки являются наследниками класса Number
- Классы-оболочки реализуют интерфейс Comparable



Пример программы 2.06.

```
/*Пример 2.06. Классы-оболочки */
public class Wrappers{
public static void main(String[] args) {

    Integer i = new Integer(100);
    Integer i1 = new Integer(i);
    Double  d = new Double(3.14);
    Float  f = new Float("3.14"); //Конструктор String
    Boolean b = new Boolean(false);

    i = i + 10;
    i++;
    d = d / 2;

    i1 = d.intValue(); //Преобразование к примитивному
}
```

- Объекты классов-оболочек имеют по умолчанию значение **null**
- Переменную базового типа можно преобразовать к соответствующему объекту, передав ее значение конструктору при объявлении объекта.

```
int i = 100;
Integer intObject = new Integer(i);
```

- Объекты класса могут быть преобразованы к любому базовому типу методами **типValue()** или обычным присваиванием

```
i = intObject.intValue();
i = intObject;
```

- Метод **valueOf(String str)** определен для всех классов-оболочек и выполняет действия по преобразованию значения, заданного в виде строки, к значению соответствующего объектного типа данных

```
Double d = Double.valueOf("1.23E+01");
```

Класс **`Character`** имеет целый ряд специфических методов для обработки символьной информации:

digit(char ch, in radix) - переводит цифру ch системы счисления с основанием radix в ее числовое значение типа int.

forDigit(int digit, int radix) - производит обратное преобразование целого числа digit в соответствующую цифру (тип char) в системе счисления с основанием radix.

toLowerCase(), touppercase(), toTitieCase() возвращают символ, содержащийся в классе, в указанном регистре.

Начиная с Java 5.0:

Автоупаковка (autoboxing) – процесс автоматической инкапсуляции данных простого типа в эквивалентный ему объект класса –оболочки

Автораспаковка (autoUNboxing) – процесс автоматического извлечения из упакованного объекта примитивного значения

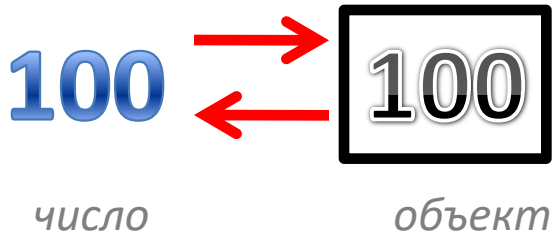
Эти процессы позволяют смешивать в выражениях значения примитивных типов и объектов оболочек, однако, требуют дополнительного времени на автоупаковку и автораспаковку.

Пример программы 2.07.

```
/*Пример 2.07. Автоупакова и автораспаковка */
public class AutoBoxing{
public static void main(String[] args) {

   Integer i = 100;  //Автоупаковка и создание объекта
   Integer j = i + 1;//Распаковка+операция+упаковка
   int k = 10;
   j = i + j + k;    //Числа и объекты
   System.out.println("i=" + i + " j=" + j + " k=" + k);
}
```

Console

```
i=100 j=211 k=10
```



Несмотря на то, что объекты классов-оболочек могут участвовать в арифметических выражениях вместе с числами, сравнивать их значения с помощью операции равенства (==) будет некорректным. Так как при этом будут сравниваться ссылки, а не значения.

Для сравнения необходимо использовать метод **`equals()`**

Пример программы 2.08.

```
/* Пример 2.08. Сравнение объектов*/
public class CompareWrappres{
public static void main(String[] args) {

  int i = 123;
  Integer i1 = new Integer(i);
  Integer i2 = new Integer(i);

  System.out.println(""+ (i == i1) + (i == i2) + (i1 == i2));
  System.out.println(""+ i1.equals(i) + i2.equals(i) +
        i1.equals(i2) );
}
```

Console

```
truetruefalse
truetruetrue
```

D. Операторы

Java содержит "стандартный" набор операторов управления, почти совпадающий с операторами языка C/C++ и подобными.

Операторы перехода	Условный оператор	**`if`**
	Оператор выбора	**`switch`**
Операторы циклов	Цикл с параметром	**`for`**
	Цикл с предусловием	**`while`**
	Цикл с постусловием	**`do..while`**
	Цикл по массивам и коллекциям	**`for`**
Операторы прерывания	Возврат из метода	**`return`**
	Прекращение цикла/перехода	**`break`**
	Прерывание итерации цикла	**`continue`**

Условный оператор

```
if (boolexp) { /*операторы1*/}
else { /*операторы2*/ }//может отсутствовать
```

- Если выражение **boolexp** принимает значение **true**, то выполняется группа операторов 1, иначе – группа операторов 2.

```
/* Условный оператор */
. . .
if(bid != null){
  item.setBestOffer(bid.getBidValue());
  item.setBid(UserDAO.getUserByID(bid.getBidID()));
}
else{
  item.setBestOffer(null);
  item.setBid(null);
};
. . .
```

Оператор выбора

```
switch(exp) {
      case exp1:/*операторы, если exp==exp1*/
            break;
      case exp2:/*операторы, если exp==exp2*/
            break;
      default: /*операторы при несовпадении*/
}
```

- При совпадении условий вида **exp==exp1** выполняются подряд все блоки операторов до тех пор, пока не встретится оператор **break**. Значения **exp1**,…, **expN** должны быть константами и могут иметь значения типа **int**, **byte**, **short**, **char** или **enum**.
- Начиная с Java 7 для выражения и констант можно использовать строковый тип и классы-оболочки.

```
/* Оператор выбора*/
. . .
switch (month)
{
case 12: case 1: case 2:
       season = "зима";
       break;
case 3: case 4: case 5:
       season = "весна";
       break;
case 6: case 7: case 8:
       season = "лето";
       break;
case 9: case 10: case 11:
       season = "осень";
       break;
default:
       season = "Нет такого месяца";
} . . .
```

Цикл с параметром

```
for(exp1; boolexp; exp3){
 /*операторы*/ }
```

- **exp1** вычисляется до начала цикла (*инициализация*)
- условие **boolexp** вычисляется перед каждой итерацией цикла; если его значение равно **true** – цикл продолжается (*предусловие*)
- **exp3** выполняется после каждой итерации (*модификация*)
- это цикл с заранее известным числом итераций

```
/* Оператор цикла с параметром*/

. . .
for (int i = 0; i < 10; i++)
      System.out.println("a[" + i + "]=" + a[i]);
. . .


for (int a = 1, b = 4; a < b; a++, b--) {
      System.out.println("a = " + a);
      System.out.println("b = " + b);
}
. . .
```

Цикл с предусловием

```
while(boolexp) {
 /*операторы*/ }
```

- пока условие **boolexp** равно **true** цикл продолжает исполняться
- это цикл с заранее неизвестным числом итераций

```
/* Оператор цикла с предусловием*/

. . .
while (a != b){
      if (a > b)
            a -= b;
      else
            b -= a;
}
. . .
```

Цикл с постусловием

```
do { /*операторы*/ }
while (boolexp);
```

- пока условие **boolexp** равно **true** цикл продолжает исполняться
- это цикл с заранее неизвестным числом итераций

```
/* Оператор цикла с постусловием*/

. . .
double a = 1.0;
double s = 0.0;
double eps = 1E-06;
do {
      a /= 2;
      s += a;
} while (a > eps);
. . .
```

Цикл по массивам и коллекциям

```
for(Тип var : exp){ /*операторы*/ }
```

- это цикл для последовательного перебора элементов массива (коллекции) **exp**
- переменная **var** последовательно принимает значения элементов
- с помощью этого цикла нельзя изменять значения элементов

```
/* Оператор цикла по массиву/коллекции */


. . .
int[] array = {1, 3, 5, 7, 9};
for (int x : array)
      System.out.printf(x + " ");
. . .
```

Возврат из метода

```
return exp;
```

- прекращает выполнение метода, возвращает в качестве его значения значение выражения **exp**
- **exp** может отсутствовать, если метод имеет тип **void**

```
/* Возврат из метода */


. . .
public static int max (int a, int b){
      return a > b ? a : b;
}
. . .
```

Прекращение цикла/перехода

```
break;
break label;
```

- прекращает выполнение оператора цикла или выбора
- если используется без метки, то прерывает ближайший объемлющий оператор цикла/перехода
- если указана **`label`**, то прекращается оператор, отмеченный меткой **`label`** (например, для вложенных циклов)

```
/* Прекращение цикла 1 */


. . .
double a = 1.0, b = 0.0;
while(true)
{
      a /= 2;
      b += a;
      if (b == 1.0)
            break;
}
. . .
```

```
/* Прекращение цикла 2*/


. . .
int k = 0;
stop:
for(int i = 0; i < 10; i++){
      for(int j = 0; j < 10; j++){
            k++;
            if (k == 50)
                  break stop;
      }
}
. . .
```

Прекращение итерации цикла

```
continue;
continue label;
```

- прекращает выполнение итерации цикла
- если используется без метки, то прерывает итерацию ближайшего объемлющего цикла
- если указана метка **`label`**, то прерывает итерацию цикла, отмеченного меткой **`label`** (для вложенных циклов)

```
/* Прекращение цикла 2*/


. . .
int s = 0;
for(int i = 0; i < 10; i++)
{
      if(i%2 == 0)
            continue;
      s += i;
}
System.out.println("s = " + s);
. . .
```

E. Класс Math

Класс **`java.lang.Math`** содержит методы и константы для математических и технических расчетов.

- Все методы класса вызываются без создания экземпляра класса

Функции пакета Math

double	abs(double a)
float	abs(float a)
int	abs(int a)
long	abs(long a)
double	acos(double a)
double	asin(double a)
double	atan(double a)
double	cbrt(double a)
double	ceil(double a)
double	copySign(double magnitude, double sign)
double	copySign(float magnitude, float sign)
double	cos(double a)
int	decrementExact(int a)
long	decrementExact(long a)
double	exp(double a)
double	floor(double a)
double	hypot(double x, double y)
double	IEEEremainder(double f1, double f2)
int	incrementExact(int a)

long	incrementExact(long a)
double	log(double a)
double	log10(double a)
double	max(double a, double b)
float	max(float a, float b)
int	max(int a, int b)
long	max(long a, long b)
double	min(double a, double b)
float	min(float a, float b)
int	min(int a, int b)
long	min(long a, long b)
int	multiplyExact(int x, int y)
long	multiplyExact(long x, long y)

int	negateExact(int a)
long	negateExact(long a)
double	pow(double a, double b)
double	random()
double	rint(double a)
int	round(double a)
double	signum(double a)
floa	signum(float a)
double	sin(double a)
double	sinh(double a)
doubl	sqrt(double a)
int	subtractExact(int x, int y)
long	subtractExact(long x, long y)
double	tan(double a)
double	tanh(double a)
double	toDegrees()
int	toIntExact(long value)
double	toRadians()

Пример программы 2.09.

```
/* Пример 2.09. Пакет Math*/
public class MathDemo{
  public static void main(String[] args) {
        final int MAX_VALUE = 10;
        double d = Math.random() * MAX_VALUE;
        System.out.println("d = " + d);
        System.out.println("Округленное до целого =" + Math.round(d));
        System.out.println("Ближайшее целое <= исходного числа =" +
               Math.floor(d));
        System.out.println("Ближайшее целое >= исходного числа =" +
               Math.ceil(d));
        System.out.println("Ближайшее целое значение к числу =" +
               Math.rint(d));
        System.out.println("sin(d) + cos(d) = " + (Math.sin(d) +
               Math.cos(d)));
  }
}
```

Console

```
d = 2.304147663648134
Округленное до целого =2
Ближайшее целое <= исходного числа =2.0
Ближайшее целое >= исходного числа =3.0
Ближайшее целое значение к числу =2.0
sin(d) + cos(d) = 0.07357210124075564
```

Задания к Теме 2:

1. Создать программу для ввода пароля и сравнения его со строкой-образцом.
2. Создать программу ввода целого числа, которая выводит разложение этого числа на простые множители.
3. Дано целое положительное число n. Найти максимальную и минимальную цифры этого числа.
4. ДЗ! Даны две стороны a и b треугольника и угол между ними alpha. Найти третью сторону и два других угла.
5. ДЗ! Дано натуральное число r, не превышающее 16. Вывести таблицу умножения в системе счисления с основанием r.