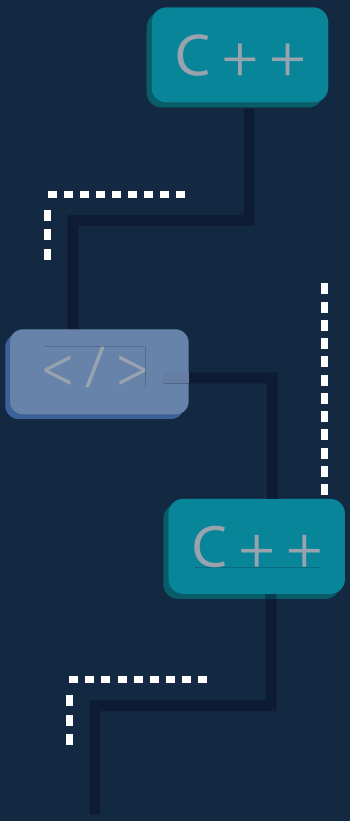

top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

Язык программирования JAVA

Урок №2

Переменные, типы данных

Содержание

1. Базовые конструкции

1.1. Понятие о строгой типизации

В Java при объявлении переменной всегда необходимо указывать ее тип. После объявления переменной она не может изменить свой тип на протяжении всей видимости переменной в коде. Строгая типизация позволяет обнаруживать ошибки в коде на этапе компиляции программы.

1.2. Типы данных

В языке Java есть две основные категории делящих типы переменных на примитивные и ссылочные.

- ***Примитивные типы*** данных (базовые типы) – это типы данных присутствующие в синтаксисе языка. Для использования примитивных типов нет необходимости создавать свои классы или использовать какие-либо библиотеки.
- ***Ссылочный тип*** (объектный тип) – это тип который в качестве значения содержит ссылку на объект.

Примитивные типы данных делятся на:

- ***целочисленные*** примитивные типы данных могут хранить в себе только целые числа (без дробной части), из диапазона заданного размерностью типа.

Таблица целочисленных типов

Наименование	Разрядность	Диапазон значений
`byte`	8 бит (1 байт)	от −128 до 127
`short`	16 бит (2 байта)	от −32 768 до 32 767

Наименование	Разрядность	Диапазон значений
int	32 бита (4 байта)	от −2147483648 до 2147483647 или от -2^{31} до 2^{31}
long	64 бита (8 байт)	от -2^{63} до 2^{63}

- ***дробные*** типы данных предназначены для хранения значений дроби. Дробный тип данных может быть представлен в виде десятичной дроби. В качестве разделителя целой и дробной части используется точка. Например 1.5 – это одна целая и пять десятых. Так же могут быть использованы научная и экспоненциальная форма записи.

Таблица дробных типов

Наименование	Разрядность	Диапазон значений
float	32 бита (4 байта)	от −3. 4e − 038.. до 3. 4e + 038
double	64 бита (8 байт)	от −1. 7e − 308.. до 1. 7e + 308

Особенности специальных значений с плавающей точкой

Выражения	Результат
Math.sqrt(-1.0)	NaN
0.0 / 0.0	NaN
1.0 / 0.0	Infinity
−1.0 / 0.0	−Infinity
NaN==NaN	false
Infinity==Infinity	false

Символьный тип может хранить в себе только целые числа, которые интерпретируются, как коды символов

из таблицы на основе юникода (*Unicode*). Юникод – эта стандарт кодирования символов, где каждому символу соответствует определенный код. В юникоде представлены все возможные знаки различных народов мира. Каждый символ кодируется двумя байтами. Если нет возможности набрать в коде символ с клавиатуры он может быть заменен на запись символа в виде юникода. Например: '\u0056'

Наименование	Разрядность	Диапазон значений
`char`	16 бит (2 байта)	от 0 до 65536

Логический тип используется для хранения значений, полученных в результате вычисления логических выражений, или может задаваться логическими литералами: где ***true*** – обозначает истину, ***false*** – обозначает ложь. В стандартной реализации «Sun JVM» для хранения значений типа boolean используется 32 бита, а в случае массива boolean происходит оптимизация до 8 бит. Для хранения большого количества значений типа boolean рекомендуется использовать класс BitSet.

Наименование	Разрядность	Диапазон значений
`boolean`	–	true или false

Преобразование типов

Java является языком со строгой (статической) типизацией и всегда контролирует работу с данными определенного типа, но, не смотря на это, Java дает программисту возможность копировать значения одного типа в другой.

В случае если размерность типа копируемого значения меньше размерности типа в который копирует-

ся, автоматически происходит ***неявное преобразование*** типа. Неявное преобразование не требует специального синтаксиса.

Пример:

```
short i = 3;
int j = i;
long n = 5; // число 5 это литерал типа int
```

Явное преобразование типов может происходить и в выражениях. Все операнды неявно преобразуются в тип, который имеет наибольшую разрядность.

Пример:

```
byte a = 37;
short b = 12;
char c = 'a';
int sum = a + b + c; // тип всех операндов повышается
      // до int
```

Таблица неявных преобразований

Исходный тип	Типы, в которые можно преобразовать неявно	Потеря точности
byte	short, int, long	Нет
short	int, long	Нет
int	long	Нет
int	float, double	Да
char	int	Нет
long	float, double	Да
float	double	Нет

Пример неявного преобразования с потерей точности:

```
int big = 1234567890;
float f = bigNumber;
System.out.printf("%f", f);
```

Результат: 1234567936.000000

Если есть необходимость скопировать значение типа, который имеет большую разрядность в тип с меньшей разрядностью, то необходимо использовать ***явное преобразование*** типов.

Синтаксис явного преобразования:

```
<тип> i = (<тип>) значение исходного типа;
```

Пример:

```
int i = (int) 7L;
float m = (float) 1.5; // 1.5 является литералом, тип
      // которого double

int j = 12;
byte b = (byte) j;
```

Ошибка явного приведения типа:

```
byte b = 55;
b = b * 2; // ошибка компиляции, так как 2 это
   // литерал типа int и результатом
   // выражения тоже будет тип int
```

При явном и неявном преобразовании типов может произойти переполнение (overflow) или потеря значений (underflow). Java никак не контролирует эти процессы и не выводит ошибки при компиляции или выполнении программы.

Пример overflow при неявном приведении:

```
byte bMax = 127;
bMax++;
System.out.println(bMax);
```

Результат: –128

Пример overflow при явном приведении:

```
short maxValue = 256;
byte bb = (byte) maxValue;
System.out.println(bb);
```

Результат: 0

Пример underflow:

```
double d1 = 0.3333333333333333;
// потеря чисел после точки, начиная с 8 знака
float f1 = (float) d1;
System.out.println(f1);
```

Результат: 0.33333334

Пример underflow:

```
float f3 = 3.64f;
int i3 = (int) f3; // дробная часть отбрасывается
System.out.println(i3);
```

Результат: 3

***Важно!!!** При явном преобразовании типов, контроль за переполнением значения или потери точности полностью ложится на плечи программиста.*

Типы-обертки (wrapper classes)

Для всех примитивных типов в Java существуют классы являющимися их обертками (оболочками) предназначенные для работы с значениями примитивных типов как с объектами (ссылочными типами).

Так как Java является объектно-ориентированным языком, то все данные, следую парадигме ООП, должны инкапсулироваться в объектах. Но использование объектов для математических вычислений существенно снижает быстродействие, усложняет код и занимает больший объем памяти. Поэтому в Java кроме ссылочных типов были введены примитивные типы, которые являются небольшим исключением из объектной модели языка. Классы обертки возвращают работу с примитивными типами к объектной модели.

Пример сложения с использованием объектов:

```
Integer sum = new Integer(2) + new Integer(2);
```

Пример сложения с примитивными типами:

```
int sum = 2 + 2;
```

Все классы обертки являются неизменными (нельзя изменить значение хранящееся внутри объекта класса-обертки), так же нельзя наследоваться от классов-оберток. Эти правила связаны с логическим предназначением классов-оберток, которые просто оборачивают числовые значения.

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

Все целочисленные типы-обертки наследуются от абстрактного класса Number.

Значения передаются в класс-обертку с помощью конструктора, в виде литерала или переменной примитивного типа или в качестве строки.

Пример:

```
Integer i = new Integer(3);
Integer i1 = new Integer("123");
Number n = new Short(4);
```

Классы-обертки имеют статические и не статические методы.

Статические методы: методы вида parseXXX преобразуют строку в значение примитивного типа.

Примеры:

```
int i1 = Integer.parseInt("123");
boolean b1 = Boolean.parseBoolean("TRUE");
```

Метод valueOf преобразует строку или число в класс-обертку.

Примеры:

```
Integer i2 = Integer.valueOf("5");
Integer i3 = Integer.valueOf(7);
Float f = Float.valueOf(1.3f);
```

Метод toBinaryString есть только у классов-оберток целых типов преобразует число в строку в бинарном представление.

Пример:

```
String binaryString = Integer.toBinaryString(21);
System.out.println(binaryString);
```

Результат: 10101

Метод toHexString преобразует число в строку в шестнадцатеричном представлении.

Пример:

```
String hexString = Float.toHexString(1.5f);
System.out.println(hexString);
```

Результат: 0x1.8p0

Пример:

```
String hexString = Integer.toHexString(255);
System.out.println(hexString);
```

Результат: ff

Класс-обертка Character имеет множество различных методов присущих только ему. Большинство из них простые, и что они делают понятно из названия методов.

Пример:

```
// проверяет, является ли символ цифрой

boolean isDigit1 = Character.isDigit('3');
boolean isUpper = Character.isUpperCase('b');
// истина, если цифра в верхнем регистре
```

> ***Совет.*** *Старайтесь всегда используйте примитивные типы, а обертки используйте только там, где невозможно использовать примитивные типы.*

Автоматическая упаковка и распаковка

Автоматическая упаковка (*autoboxing*) – это механизм неявного создания (без использования оператора **new**) объекта класса-обертки соответствующего примитивного типа.

Пример:

```
Integer i = 5;
Boolean b = true;
```

Автоматическая распаковка (unboxing) – это механизм неявного преобразования объекта типа-обертки в примитивный тип соответствующего типа.

Пример:

```
int i = new Integer(5);
```

Автоматическая упаковка и распаковка может происходить при присвоении или при передаче в метод.

Пример автоупаковки:

```
public static void main(String[] args)
{
 test(3); // литерал 3 упаковывается в объект
    // класса-обертки
}
public static void test(Integer value)
{
 System.out.println(value);
}
```

Автоматическая упаковка и распаковка упрощают код, но могут привести к ряду ошибок, которые можно выявить только на этапе выполнения программы.

Пример:

```
Integer i = null;
int j = i; // возникнет исключение NullPointerException
```

Пример:

```
Number n = 555;
byte b = (Byte) n; // возникнет исключение
     // ClassCastException
```

Пример:

```
static void inc(List<Integer> list, int v) {
 for (Integer i:list) {
  // здесь не происходит изменение значения
  // переменной в списке
  i += v;
 }
}
```

Примечание. *Autoboxing и unboxing появились в Java с версии JDK 1.5.*

1.3. Комментарии

Язык Java позволяет вставлять в исходный код различные комментарии, замечания и пояснения, которые не подвергаются компиляции и не попадают в байт-код.

- `//` – строчный комментарий, все что находится после двойных обратных черт до конца строки не будет скомпилировано.

Пример:

```
int i = 5; // int i = 3;
```

Блочный комментарий позволяет выделить часть исходного кода, которая будет исключена из компиляции:

- `/*` — начало блочного комментария,
- `*/` — конец блочного комментария.

Пример:

```
/*
  Пример блочного комментария
  int i =5;
*/
i = 3; // ошибка компиляции, так как переменная i
  // не объявлена в коде, а находится внутри блока
  // комментариев.
```

javadoc – особый вид комментариев предназначенный для описания назначения классов, методов и полей классов. В комплект JDK входит утилита, которая собирает в документ (html) все комментарии имеющие тип javadoc по всем файлам исходного кода проекта и на их основе создает html документ API (*application programming interface*).

Пример:

```
/**
 * Описание назначения класса
 * @see Config (вместе с этим смотри класс Config)
 * @author VUnguryan (автор класса)
 * @ since 1.2.1 (версия класса)
 */
public class Controller {
}
```

Пример API для Java классов.

1.4. Переменные

Переменная – это именованная область памяти, в которую может быть записано или перезаписано и откуда может быть прочитано значение определенного типа.

Тип переменной и ее имя (идентификатор) задаются в момент объявления переменной в программе.

Синтаксис объявления переменной:

```
тип идентификатор;
```

Пример:

```
int a;
byte b1;
boolean $_$;
```

Синтаксис языка позволяет объявить сразу несколько переменных одного типа в одной строке, но так не рекомендуется делать, по Java Code Conventions.

Пример:

```
float x, y, z;
```

Переменные объявленные внутри метода называются ***локальными переменными*** и перед непосредственным использованием требуют явной инициализации.

Видимость переменной в коде ограничена местом ее объявления (декларации) и блочным разделителем (фигурными скобками). Переменная объявленная внутри тела класса называется полем класса, в случае отсутствия инициализации переменная экземпляр принимает значение по умолчанию (в зависимости от типа).

Пример:

```
class A
{
 static int x; // поле класса
 public static void main(String[] args) {
  int y; // локальная переменная
  System.out.println(y);// результат ошибка
        // компиляции с сообщением
  System.out.println(x); // результат 0
 }
}
```

Имя переменной:

- может содержать все символы латинского алфавита, цифры, знак $ и знак _;
- не может начинаться с цифры;
- не может совпадать с ключевыми словами Java.

Примеры правильного именования переменных:

```
long startTime;
int x;
boolean flag;
```

Примеры не правильного именования переменных:

```
int 1side;
char goto;
long st@rt;
```

Использование символов $ и _ в имени переменных считается плохой практикой.

Переменные принято именовать согласно рекомендациям документа Java Code Conventions.

1.5. Константы

Константа – это переменная которая должна быть инициализирована в месте объявления (или в конструкторе класса) и не может изменять своего значения на протяжении видимости этой переменной в коде. Для обозначения константы используется ключевое слова ***final***.

Пример:

```
final int const = 5;
const = 3; // ошибка компиляции
final int const1; // ошибка компиляции
```

Под понятие константы подходит понятие литерала.

Литерал – это явно заданное в коде значение определенного типа. Для задание литералов в коде программы используется специальный синтаксис, см. таблицу.

Тип литерала	Тип данных	Спец-символы	Пример
Целочисленный	int	x	3, 03, 0x3
	long	l, L	3l, 3L
Дробный	float	f, F, e	1.5f, 1.5F, 1.5e-1f
	double	d, D, e	1.5, 1.5d, 1.5D, 1.5e-1
Булевый	boolean	true, false	true, false
Символьный	char	'	'a', '\u0041'
Ссылочный	Все ссылочные	null	null

Примечание. *Начиная с версии JDK 7.0, Java позволяет разделять разряды числовых литералов символом подчеркивания. Например: 100_000_000.*

Не рекомендуется использовать маленькую букву l для обозначения литерала типа long, так как возможно неверное толкование (похоже на цифру 1).

1.6. Ввод-вывод в консольной программе

Для вывода данных на консоль необходимо использовать класс System и его статическую переменную out.

Примеры:

```
System.out.println(«text»);// вывод текста с переходом
       // на новую строку
System.out.println(5); // вывод литерала с переходом
      // на новую строку
```

```
int i = 6;
// вывод значения переменной
System.out.println(i);
// вывод текста без перехода на новую строку
System.out.print(«text»);
// форматированный вывод переменных
System.out.printf("x = %f", 0.5f);
```

Для ввода данных с консоли необходимо использовать класс Java.util.Scaner. Класс Scaner имеет ряд методов, позволяющих работать с потоком ввода.

Класс Scaner представляет из себя поток данных, получаемых из источника.

Когда используется класса из библиотеки или из другого пакета необходимо указать, откуда его загружать. Для этого в начале файла до тела класса необходимо добавить :

```
import Java.util.Scanner;
```

Пример:

```
public static void main(String[] args)
{
 // создаем объект класса сканер и передаем
 // в конструктор стандартный поток вывода
 Scanner scanner = new Scanner(System.in);
 System.out.println("Введите число");
 // возвращает истину, если введен целое число
 boolean isInt = scanner.hasNextInt();
 System.out.println(isInt);
 // проверяем, являлось ли введенное значение
 // целым числом
 if (isInt)
 {
```

```
  // считываем из консоли значение целого числа
  // в переименую x
  int x = scanner.nextInt();
  System.out.println(«Вы ввели число « + x);
 }
 else
 {
  System.out.println(«Это не целое число»);
 }
}
```

System.in – стандартный поток ввода с клавиатуры.