TECHIN

02 Duomenų tipai, kintamieji

Jaroslav Grablevski

Turinys

- Komentarai;
- Kintamieji;
- Operatoriai;
- Duomenų įvedimas;
- Matematikos klasė Math;

Basic terminology

- **Program –** a sequence of instructions (called statements), which are executed one after another in a predictable manner. Sequential flow is the most common and straightforward sequence of statements, in which statements are executed in the order that they are written – from top to bottom in a sequential manner;
- **Statement –** a single action (like print a text) terminated by semi-colon (;);
- **Block –** a group of zero, one or more statements enclosed by a pair of braces {...}; There are two such blocks in the program above.
- **Method –** a sequence of statements that represents a high-level operation (also known as subprogram or procedure).
- **Identifier or name –** a word that refers to something in a program (such as a variable or a function name);

Komentarai | Comments (1)

- Programos tekstą padeda suprasti komentarai, kurie skirti programuotojui ir visai neturi įtakos programos vykdytojui.
- Java kalboje yra 3 skirtingi komentarų tipai:
  - Vienos eilutės komentaras;
  - Kelių eilučių komentaras;
  - JavaDoc komentaras.

Komentarai | Comments (2)

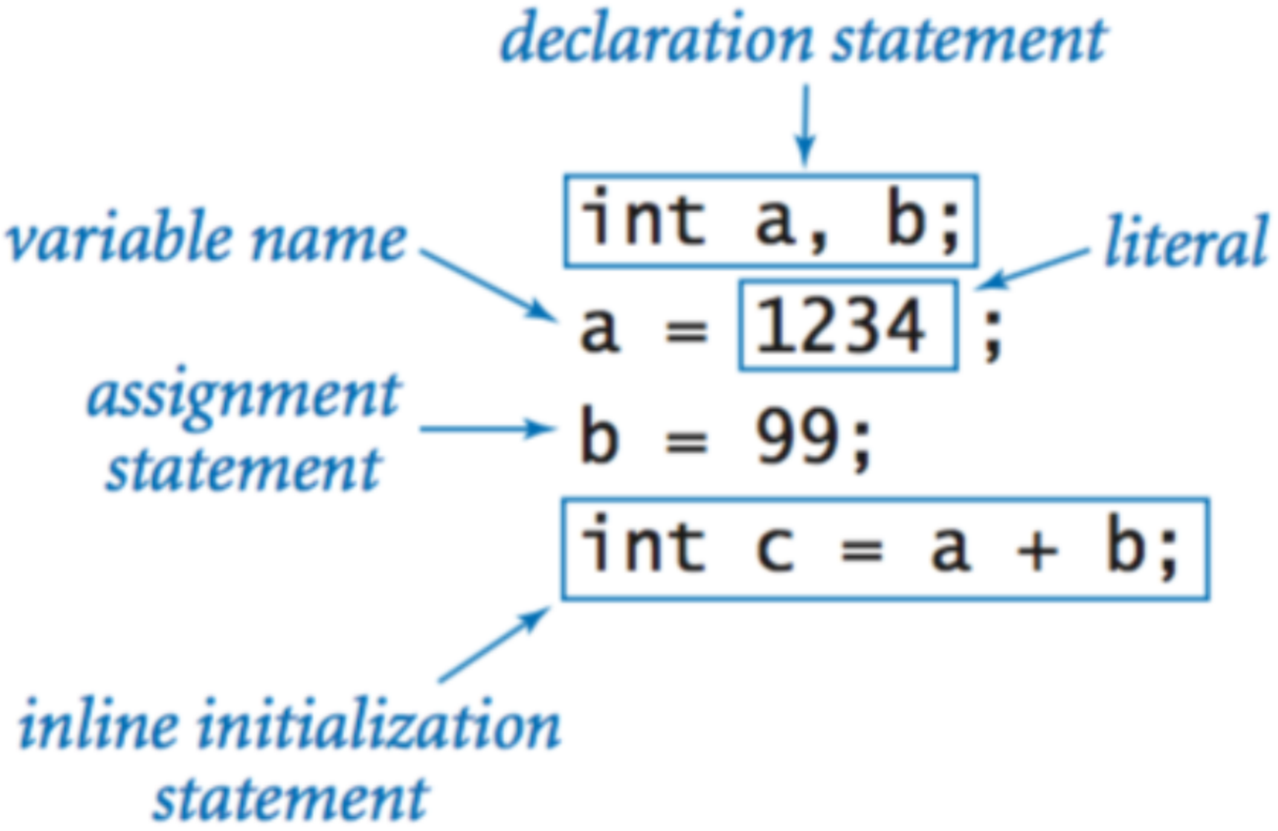
```java
/**
 * The main method accepts an array of string arguments
 *
 * @param args from the command line
 */
public static void main(String[] args) {

    // The line below will be ignored
    // System.out.println("Hello, World");

    // This is single-line comment
    System.out.println("Hello, Java"); // Here can be any comment

    /*  This is an example of
    a multi-line comment */
}
```

Kintamieji | Variables

- Kintamieji skirti pradinių duomenų reikšmėms saugoti;
- Kintamasis turi tipą, vardą ir reikšmę;
  - Tipas – *int, double, char, String, ...;*
  - Vardas – *a, plotas, raide, pirmaZinute, ...;*
  - Reikšmė – *5, 2.99, ‘A‘, “Labas“, ...;*

```
int a = 5;
double plotas = 2.99;
char raide = 'A';
String pirmaZinute = "Labas";
```

Kintamieji | Variables



Kintamieji | Variables

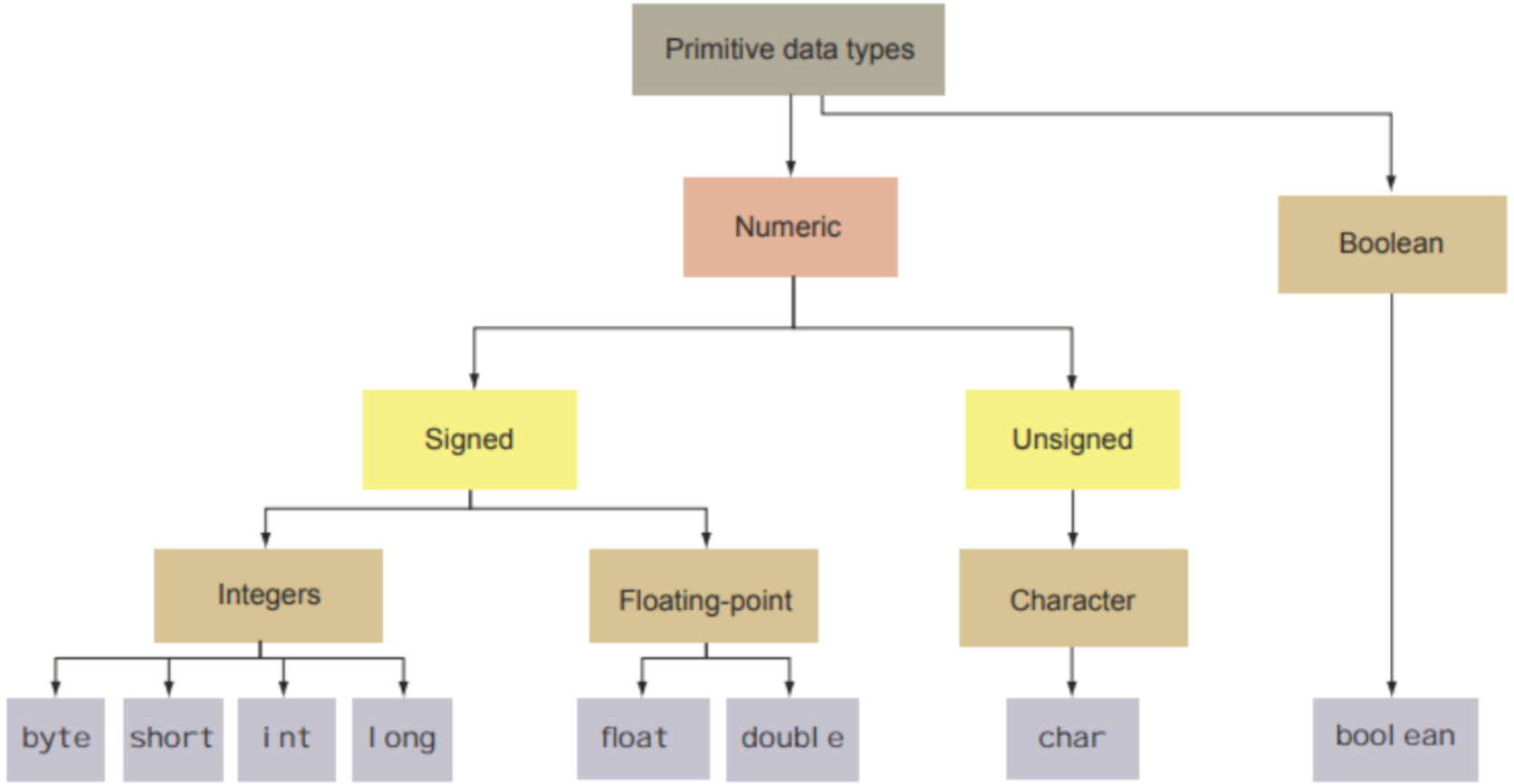
```java
int one = 1;
int num = one;

String language = "java", version = "8 or newer";

int age; // declaration
age = 35; // initialization

age = 36; //ok
age = "37"; //it does not work!
```

Var (type inference)

- Since Java 10, you can write var instead of a specific type to force automatic type inference based on the type of assigned value:

```
var language = "Java"; // String
var version = 10; // int

language = 10; // error
```

Primityvūs duomenų tipai



Primityvūs duomenų tipai

TYPE NAME	KIND OF VALUE	MEMORY USED	SIZE RANGE
`boolean`	`true` or `false`	1 byte	Not applicable
`char`	Single character (Unicode)	2 bytes	Common Unicode characters
`byte`	Integer	1 byte	−128 to 127
`short`	Integer	2 bytes	−32768 to 32767
`int`	Integer	4 bytes	−2147483648 to 2147483647
`long`	Integer	8 bytes	−9223372036854775808 to 9223372036854775807
`float`	Floating-point number	4 bytes	$\pm 3.40282347 \times 10^{+38}$ to $\pm 1.40239846 \times 10^{-45}$
`double`	Floating-point number	8 bytes	$\pm 1.76769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$

Primityvūs duomenų tipai

```
byte b = 1;
short s = 100;
int i = 1000;
long l = 1000000L;

double pi = 3.1415;
float e = 2.71828f;

char letter = 'a';

boolean enabled = true;
```

Gale „f“ raidę pridėti būtina - taip nurodome kad tai float (pagal nutylėjimą double)

Integer literals

- Java has only two types of integer literals: int and long

```
//int literals
byte age = 46;
short height = 165;
int max = 2147483647;

//long literals
long bigNum = 2147483648L;
long smallNum = 5L;
int num = 5L; // error
long tooBigInt = 2147483648; // error, too big integer literal
```

Reikia nurodyti gale raidę L, nes kitaip traktuojama kaip int (pagal nutylėjimą)

Integer literals (binary, octal, hexadecimal)

```
// no prefix --> decimal literal
int dec = 110;
int dec2 = 110_755_680; // = 110755680

// '0b' prefix --> binary literal
int binValue = 0b110110; // = 2^5*1+2^4*1+2^3*0+2^2*1+2^1*1+2^0*0 = 54
int decValueB = 110110;  // = 110110

// '0' prefix --> octal literal
int octValue = 027; // = 8^1 * 2 + 8^0 * 7 = 23
int decValue = 27; // = 27

// '0x' prefix --> hexadecimal literal
int hexValue1 = 0x29;   // 16^1 * 2 + 16^0 * 9 = 41
int hexValue2 = 0x4B;   // 16^1 * 4 + 16^0 * 11 = 75
```

Floating-point literals

```
double num1= 123.723; // a double literal
float  num2= 34.0F;   // a float literal

float num3 = 0.05;    // error!

//Scientific notation

float num4 = 0.05e3F; // = 0.05 * 10^3 = 50
double num5 = 25.255e-4;  // = 25.255 * 10^(-4) = 0.0025255
```

Char literals and escape sequences

```
char ch = 'A';
char dollar = '$';
char space =' ';
char uniChar = '\u0041'; //A
char num = 65;  //A

//Escape sequences
char newLine = '\n';
char tab = '\t';
char singleQuote ='\'';
char backslash = '\\';
```

Escape character'ius naudosite ir su String

Nurodant skaičių, reikšmė bus paimta iš ASCII lentelės. Dec

Kintamųjų vardai | Variable names

- Kintamojo vardui užrašyti galime naudoti
  - Lotyniškas raides (a-z, A-Z);
  - Skaičius (0-9);
  - Pabraukimo simbolį (_);
  - Dolerio simbolį ($);
- Kintamojo vardas privalo prasidėti raide (a-z, A-Z) arba pabraukimo simboliu (_);
- Pirmasis simbolis negali būti skaitmuo;
- Pavyzdžiai:
  x1, y1, size, roomNumber, xMax, y_Max

Tai vadinasi camelCase;
įprastai kintamuosius, metodus vadinsite pagal tai

Kintamieji | Variables

Properties of valid Identifiers	Properties of invalid identifiers
Unlimited length	Same spelling as a Java reserved word or keyword (see table 2.8)
Starts with a letter (a–z, upper- or lowercase), a currency sign, or an underscore	Uses special characters: `!`, `@`, `#`, `%`, `^`, `&`, `*`, `(`, `)`, `'`, `:`, `;`, `[`, `/`, `\`, `}`
Can use a digit (not at the starting position)	Starts with a Java digit (`0–9`)
Can use an underscore (in any position)	
Can use a currency sign (in any position): `$`, `£`, `¢`, `¥` and others	
Examples of valid identifiers	**Examples of invalid identifiers**
`customerValueObject`	`7world` (identifier can't start with a digit)
`$rate, £Value, _sine`	`%value` (identifier can't use special char `%`)
`happy2Help, nullValue`	`Digital!, books@manning` (identifier can't use special char `!` or `@`)
CONSTANT - pvz, PI, DAYS_IN_JULY	`null`, `true`, `false`, `goto` (identifier can't have the same name as a Java keyword or reserved word)

Raktiniai rezervuoti Java žodžiai

Java keywords and reserved words that can't be used as names for Java variables

abstract	default	goto	package	this
assert	do	if	private	throw
boolean	double	implements	protected	throws
break	else	import	public	transient
byte	enum	instanceof	return	true
case	extends	int	short	try
catch	false	interface	static	void
char	final	long	strictfp	volatile
class	finally	native	super	while
const	float	new	switch	
continue	for	null	synchronized	

[List of Java keywords]

Kintamųjų išvedimas (1)

```
String text = "includes text";
int wholeNumber = 123;
double decimalNumber = 3.141592653;

System.out.println("text value is " + text);
System.out.println("wholeNumber value is  " + wholeNumber);
System.out.println("decimalNumber value is " + decimalNumber);
```

```
text value is includes text
wholeNumber value is  123
decimalNumber value is 3.141592653
```

Kintamųjų išvedimas (2)

```
int wholeNumber;
wholeNumber = 123;
System.out.println("wholeNumber value is  " + wholeNumber);

wholeNumber = 42;
System.out.println("wholeNumber value is  " + wholeNumber);
```

```
wholeNumber value is  123
wholeNumber value is  42
```

TIK pirmą kartą aprašant kintamąjį nurodomas jo tipas!

Kintamųjų išvedimas (3)

```
int x = 5;
int y = -9;

System.out.println("x = " + x + ", y = " + y);
```

```
x = 5, y = -9
```

Java operatoriai

- List of Java operators in the order of precedence

postfix increment and decrement	`++ --`
prefix increment and decrement, and unary	`++ -- + - ~ !`
multiplicative	`* / %`
additive	`+ -`
bit shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Aritmetiniai operatoriai (1)

```
int first = 9;
int second = 4;
int sum, sub, mul, div, mod;

sum = first + second;
sub = first - second;
mul = first * second;
div = first / second;
mod = first % second;

System.out.println("sum = " + sum);
System.out.println("sub = " + sub);
System.out.println("mul = " + mul);
System.out.println("div = " + div);
System.out.println("mod = " + mod);
```

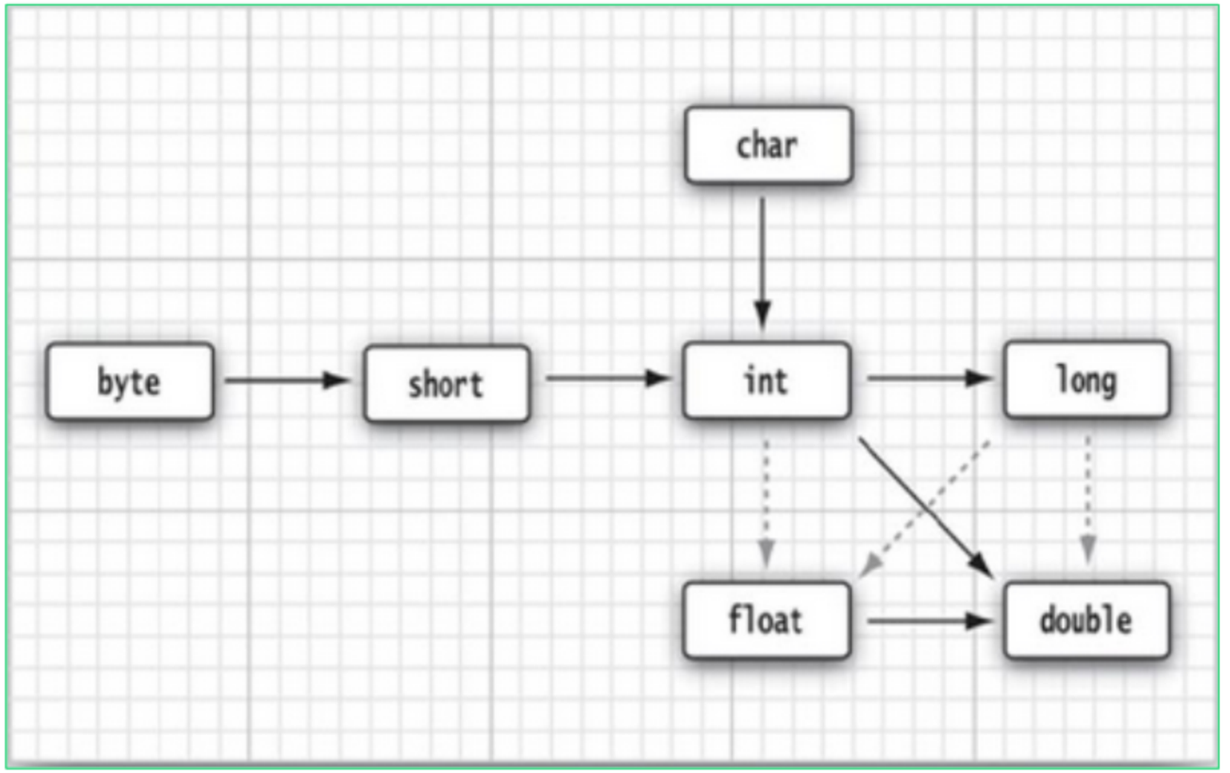
```
sum = 13
sub = 5
mul = 36
div = 2
mod = 1
```

Aritmetiniai operatoriai (2)

```
int first = 9;
int second = 4;
int div, mod;
double d1, d2, d3;

div = first / second; // 2
mod = first % second; // 1

d1 = first / second;                // 2.0
d2 = first / (double) second;       // 2.25
d3 = (double) first / second;       // 2.25
```

Type conversions

- Solid lines indicate conversions that will not lose precision in the numbers
- Dashed lines will lose some precision if converted



Implicit casting (1)

```java
//from int to long
int num = 100;
long bigNum = num; // 100L

//from long to double
long longNum = 100_000_000L;
double bigFraction = longNum; // 100000000.0

//from short to int:
short shortNum = 100;
int intNum = shortNum; // 100
```

Implicit casting (2)

```
//from char to int
char character = 'a';
char upperCase = 'A';
int ascii1 = character; // this is 97
int ascii2 = upperCase; // this is 65

//loss of precision
long bigLong =  1_200_000_002L;
float bigFloat = bigLong; // 1.2E9 (= 1_200_000_000)
```

Explicit casting

```java
double d = 2.00003;

// it loses the fractional part
long l =  (long) d; // 2

// requires explicit casting because long is wider than int
int i = (int) l; // 2

// requires explicit casting because the result is long (indicated by L)
int val = (int) (3 + 2L); // 5

// casting from a long literal to char
char ch = (char) 55L; // '7'

//type overflow !!!
long bigNum = 100_000_000_000_000L;
int n = (int) bigNum; // 276447232 - The value has been truncated!
```

Duomenų įvedimas

```java
import java.util.Scanner;

public class PirmojiPrograma {
    public static void main(String[] args) {

        Scanner reader = new Scanner(System.in);

        System.out.print("Type a word(String): ");
        String word = reader.nextLine();

        System.out.print("Type a number(integer): ");
        int numberInt = Integer.parseInt(reader.nextLine());

        System.out.print("Type a number(double): ");
        double numberDouble = Double.parseDouble(reader.nextLine());

        reader.close();

        System.out.println(numberInt + " " + numberDouble + " " + word);
    }
}
```

Matematikos klasė *Math* (1)

- Standartinė Java matematikos biblioteka;
- Biblioteką (klasę) aprašo elementarias matematines operacijas.
- Turi dvi konstantas: eulerio (E) ir pi (PI);

Matematikos klasė *Math* (2)

```java
int a = -9;
int absolute = Math.abs(a); // 9

double b = 36;
double result = Math.sqrt(b); // 6

double x1 = 5.7, y1 = 8.99;
double maxResult = Math.max(x1, y1); // 8.99

double x2 = 2.3, y2 = -5.87;
double minResult = Math.min(x2, y2); // -5.87

double pi = Math.PI;

System.out.println(absolute + " " + result);
System.out.println(maxResult + " " + minResult + " " + pi);
```

Final variable

- The only difference between a regular variable and a final variable is that we cannot modify the value of a final variable once assigned.

```
final double PI = 3.1415;
final String HELLO_MSG = "Hello";

System.out.println(PI); // 3.1415
System.out.println(HELLO_MSG); // Hello

PI = 3.1416; // error line
```

Infinity, NaN

```java
double posInf = Double.POSITIVE_INFINITY;  // +Infinity

double anotherPosInf = +1 / 0.0;           // it's +Infinity, not an exception

double posInfAgain = anotherPosInf + 100; // +Infinity again

double negInf = Double.NEGATIVE_INFINITY; // -Infinity

double negInfAgain = negInf * 100;        // The result is -Infinity

double anotherNegInf = -1.002 / 0.0;      // It is also -Infinity

double nan = Double.NaN;                  // the NaN constant

double anotherNan = 0 / 0;    // it's the NaN, not an exception

int ex = 0 / 0;     //exception
```

In Java, Infinity (∞, -∞) occurs in double and float when dividing by zero or exceeding Double.MAX_VALUE, while NaN represents undefined results (like 0.0/0.0), but integers (int, long) lack these because they use fixed-size two's complement representation and wrap around on overflow instead.