



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

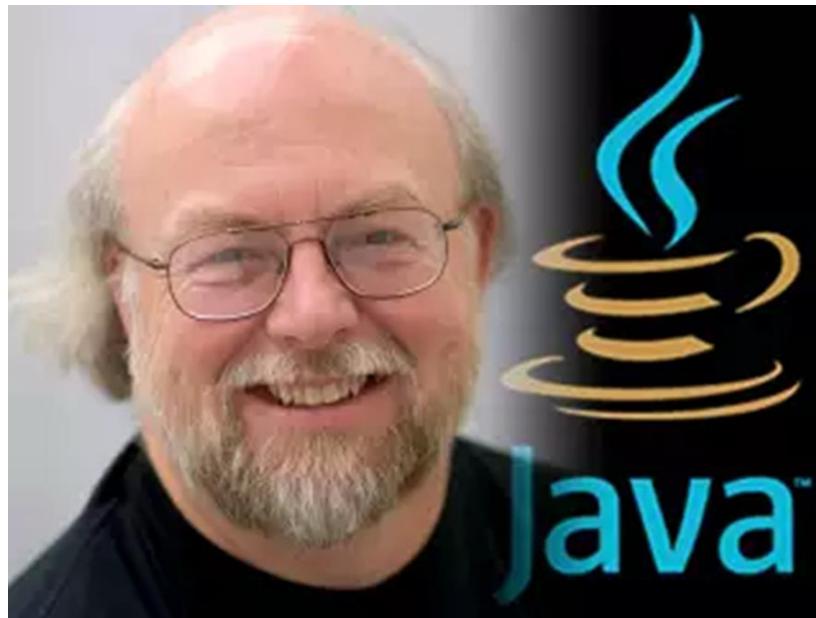
01 – NPĮK – Istorija, sąvokos

Jaroslav Grablevski

Turinys

- Istorija
- Kodėl java?
- Sąvokos
- Integruotos kūrimo aplinkos

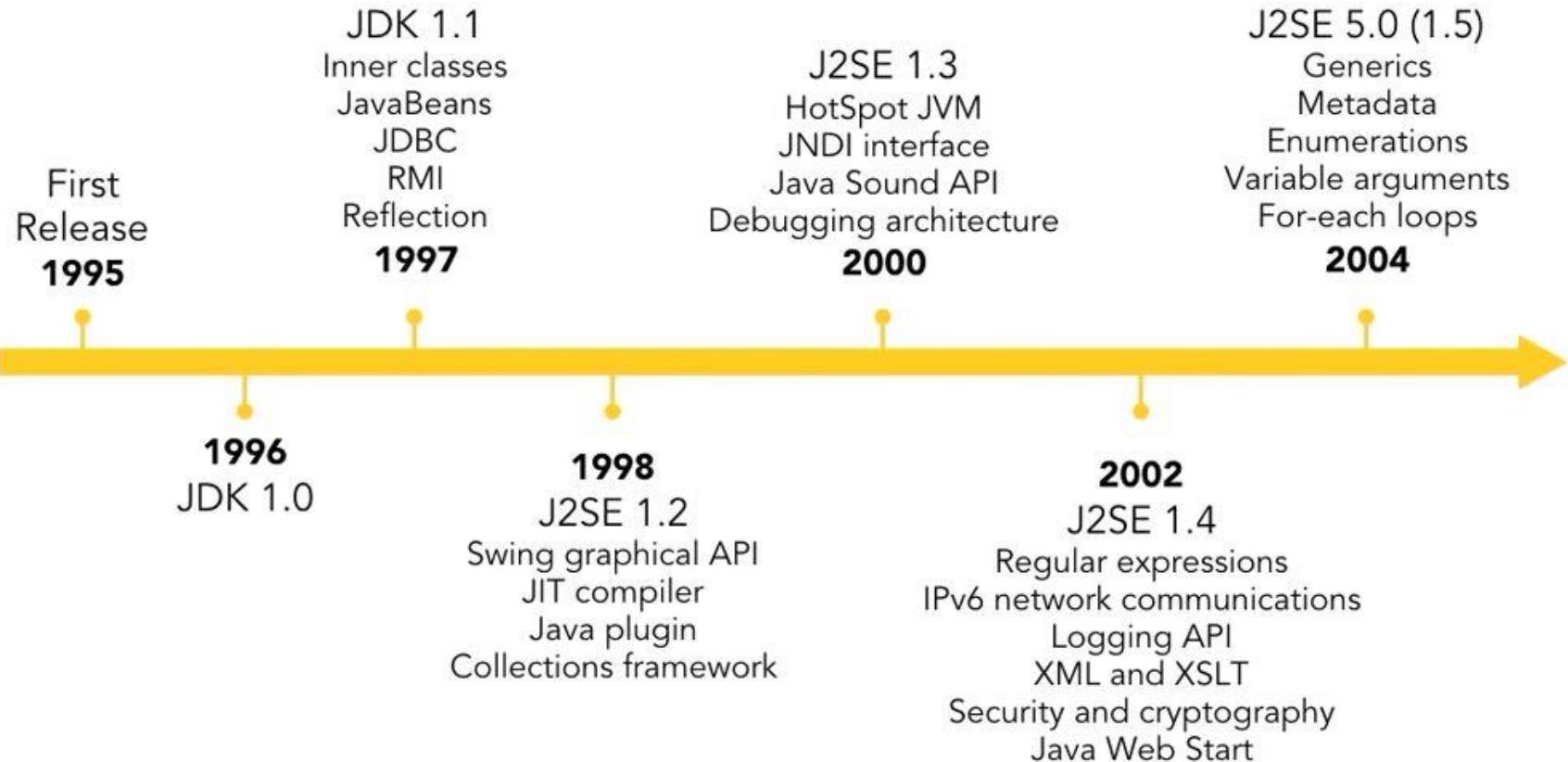
Istorija



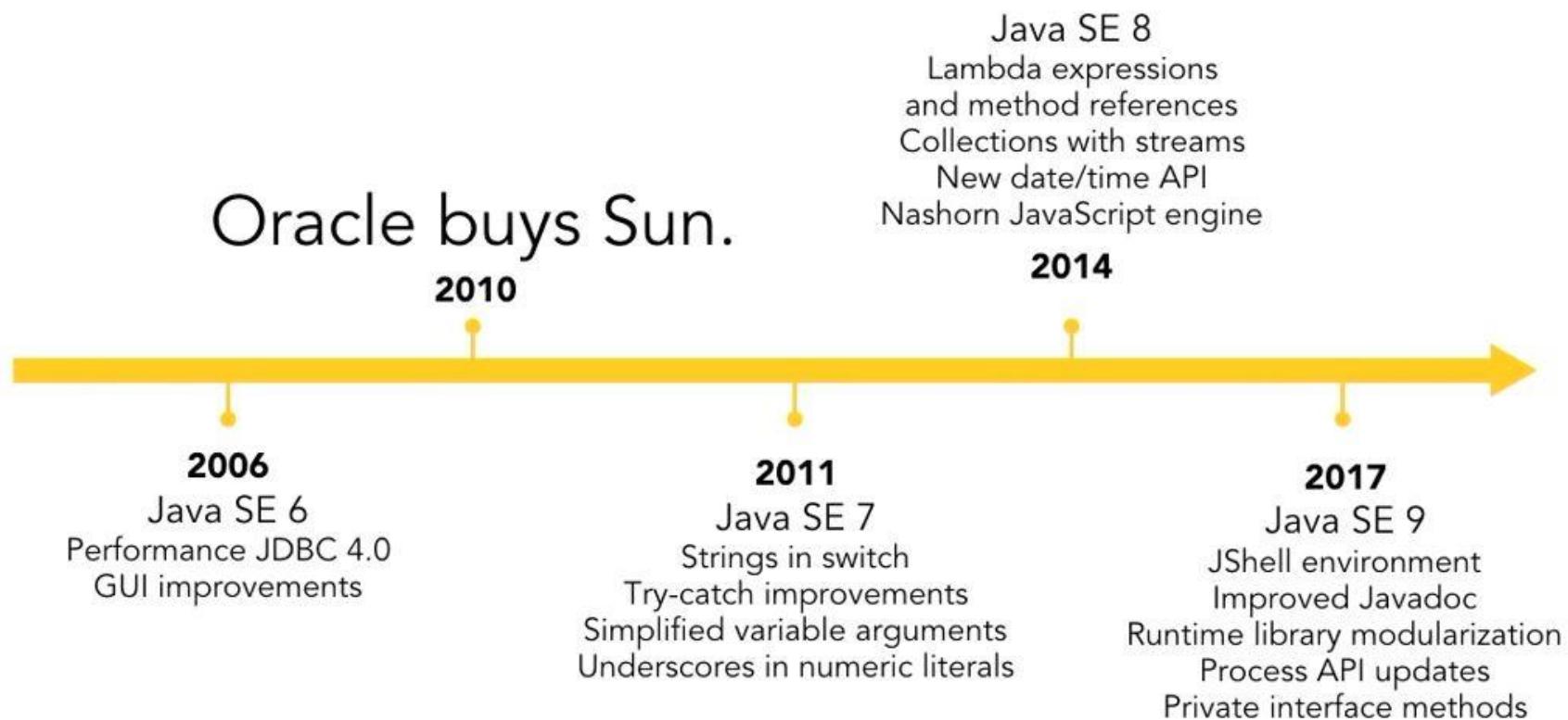
the
green
PROJECT



Java versijos (1)



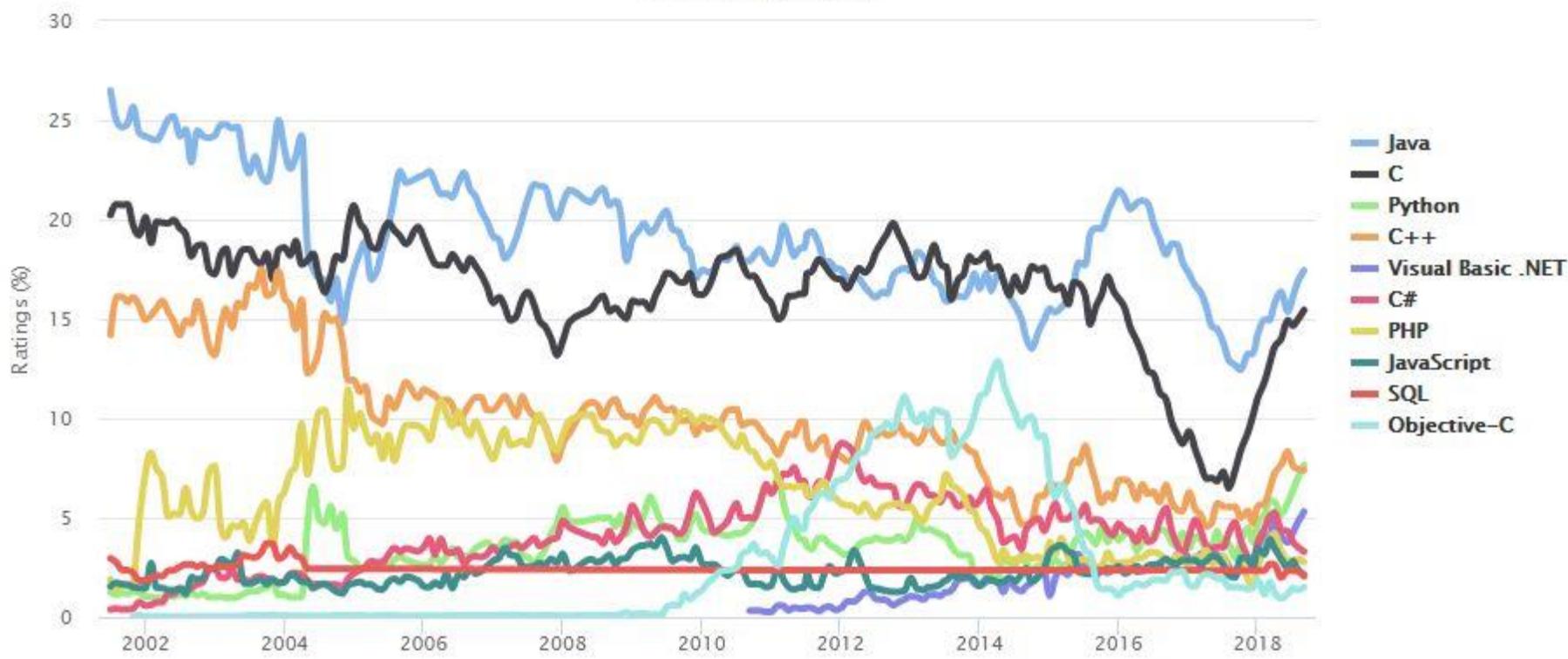
Java versijos (2)



Kodél JAVA

TIOBE Programming Community Index

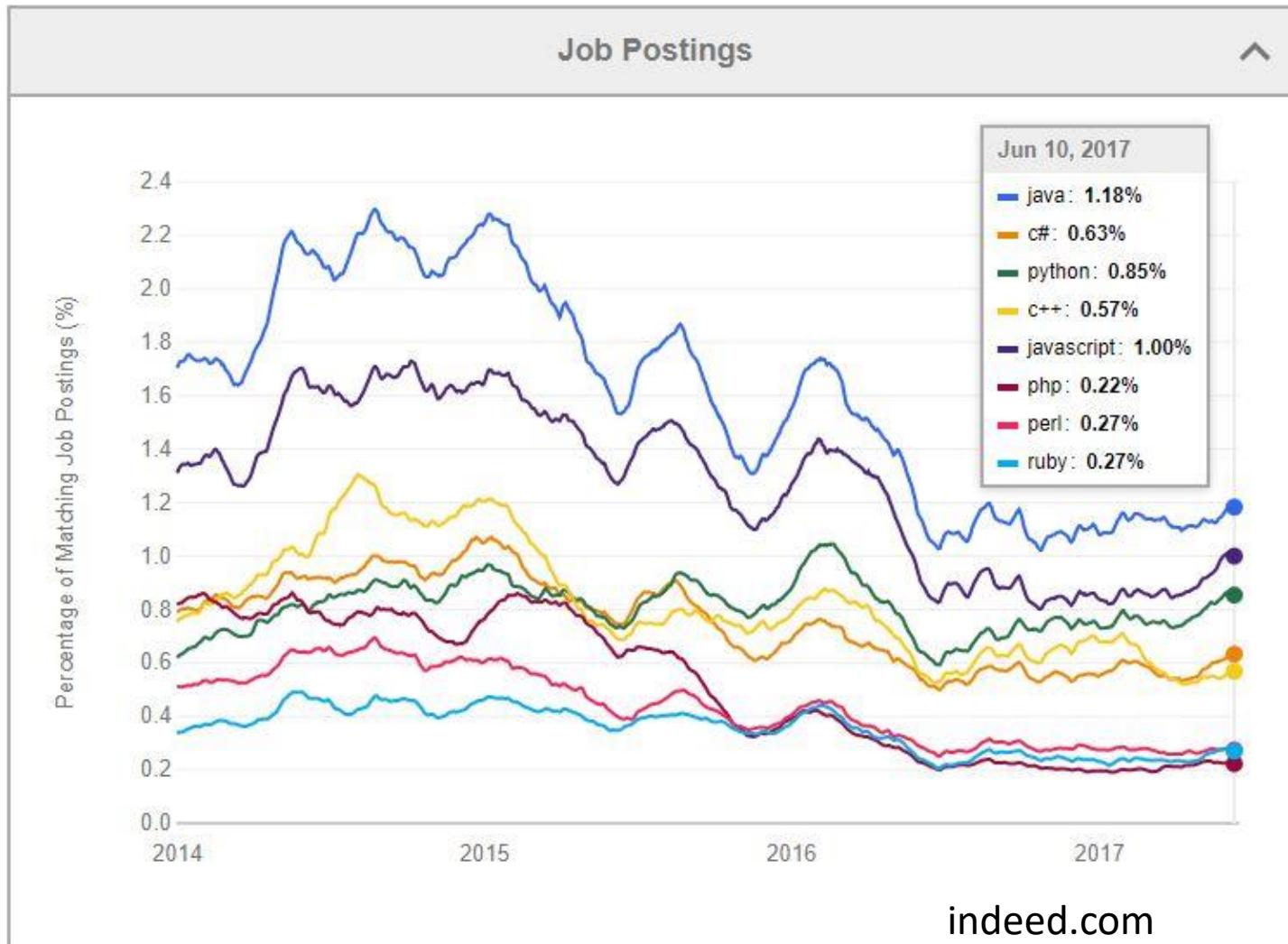
Source: www.tiobe.com



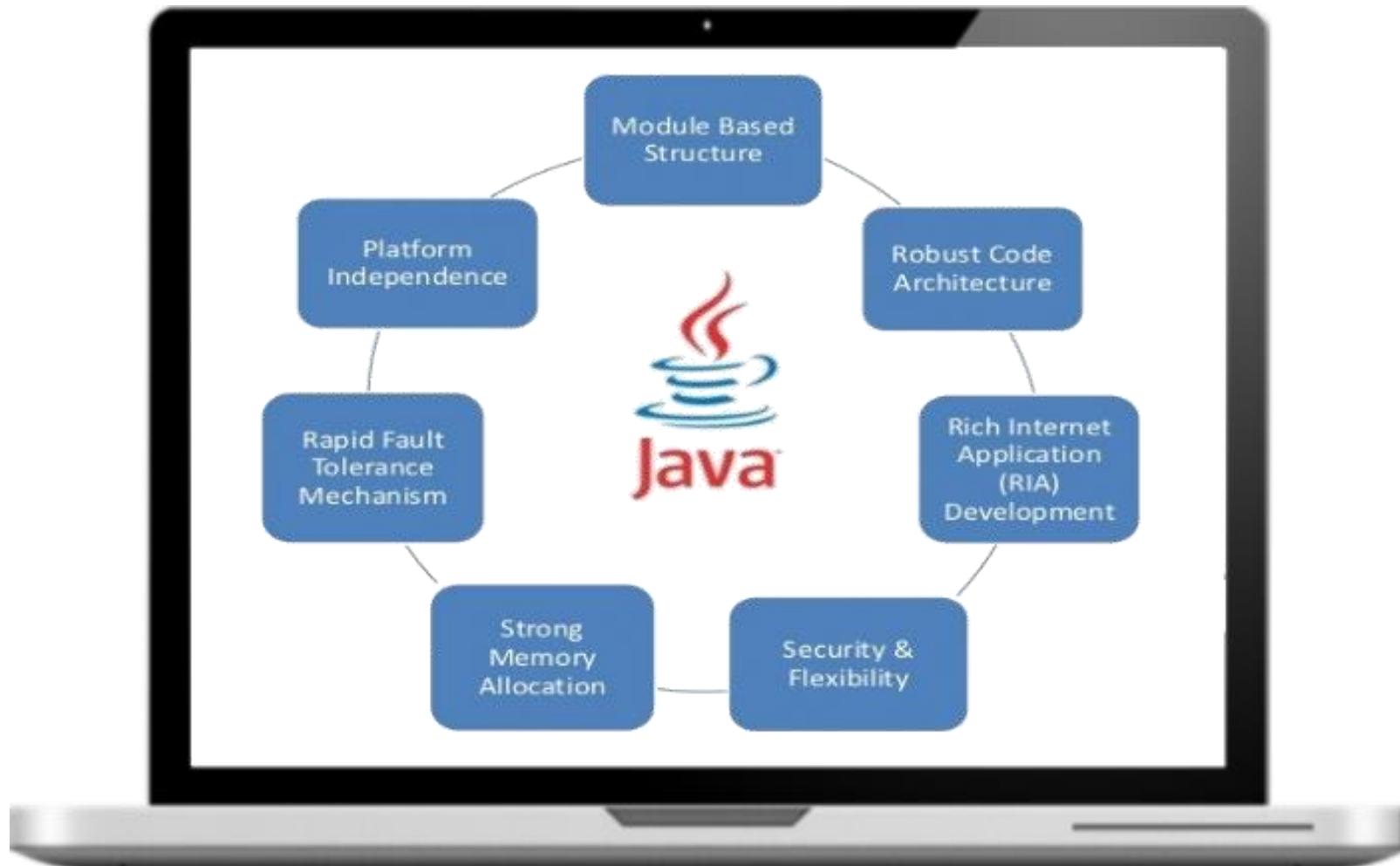
Kodél JAVA

Sep 2018	Sep 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.436%	+4.75%
2	2		C	15.447%	+8.06%
3	5	▲	Python	7.653%	+4.67%
4	3	▼	C++	7.394%	+1.83%
5	8	▲	Visual Basic .NET	5.308%	+3.33%
6	4	▼	C#	3.295%	-1.48%
7	6	▼	PHP	2.775%	+0.57%
8	7	▼	JavaScript	2.131%	+0.11%
9	-	▲	SQL	2.062%	+2.06%
10	18	▲	Objective-C	1.509%	+0.00%
11	12	▲	Delphi/Object Pascal	1.292%	-0.49%
12	10	▼	Ruby	1.291%	-0.64%
13	16	▲	MATLAB	1.276%	-0.35%
14	15	▲	Assembly language	1.232%	-0.41%
15	13	▼	Swift	1.223%	-0.54%
16	17	▲	Go	1.081%	-0.49%
17	9	▼	Perl	1.073%	-0.88%

Kodél JAVA

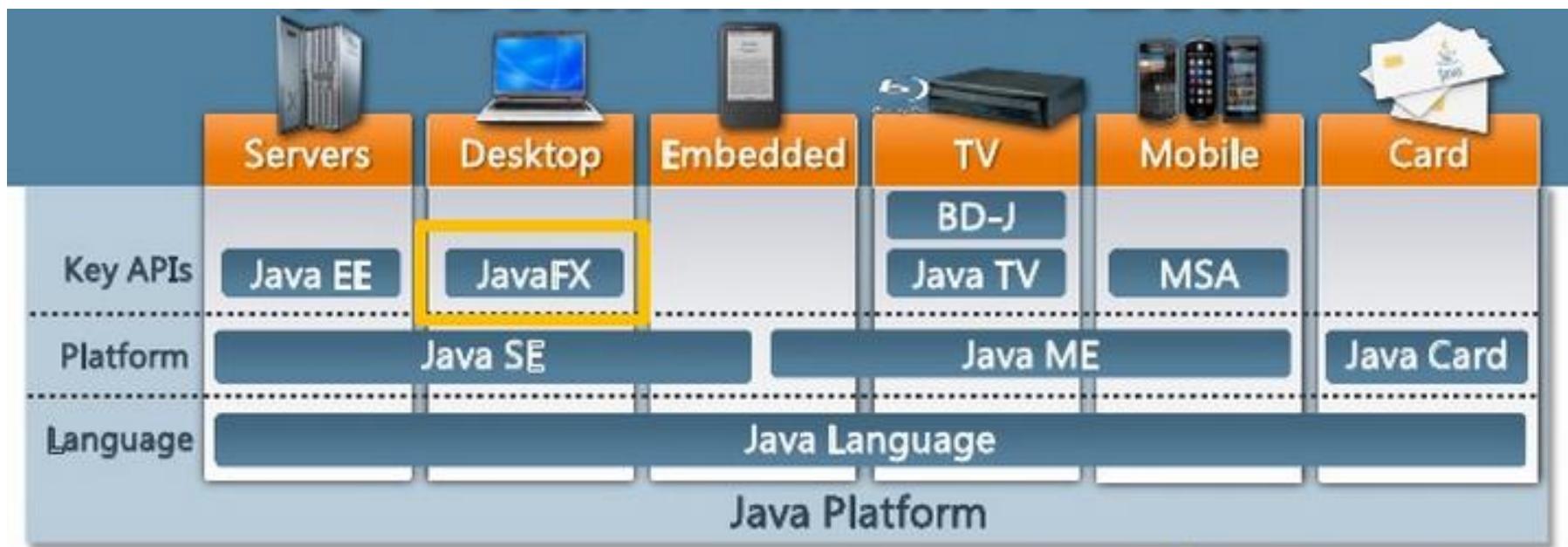


Kodél JAVA

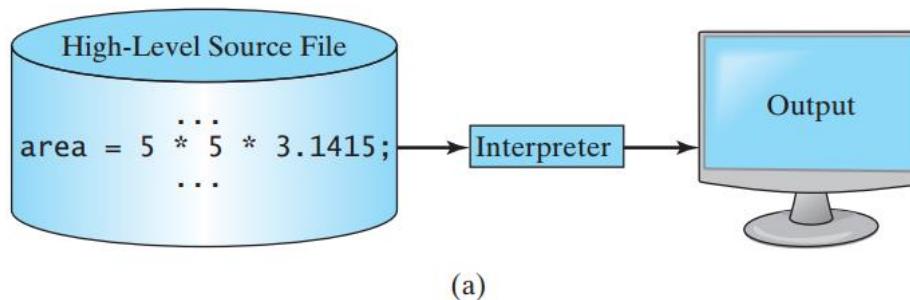


Java redakcijos

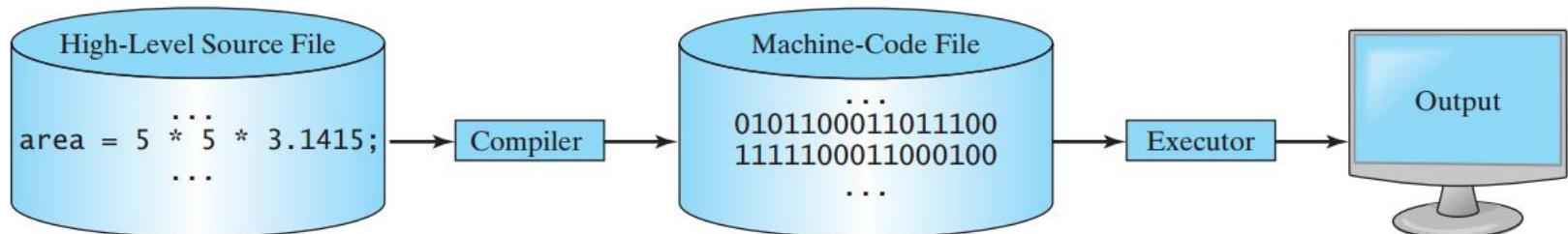
- Java SE - Standart Edition
- Java EE - Enterprise Edition
- Java ME - Micro Edition



Kompiliuojamos ir interpretuojamos programavimo kalbos

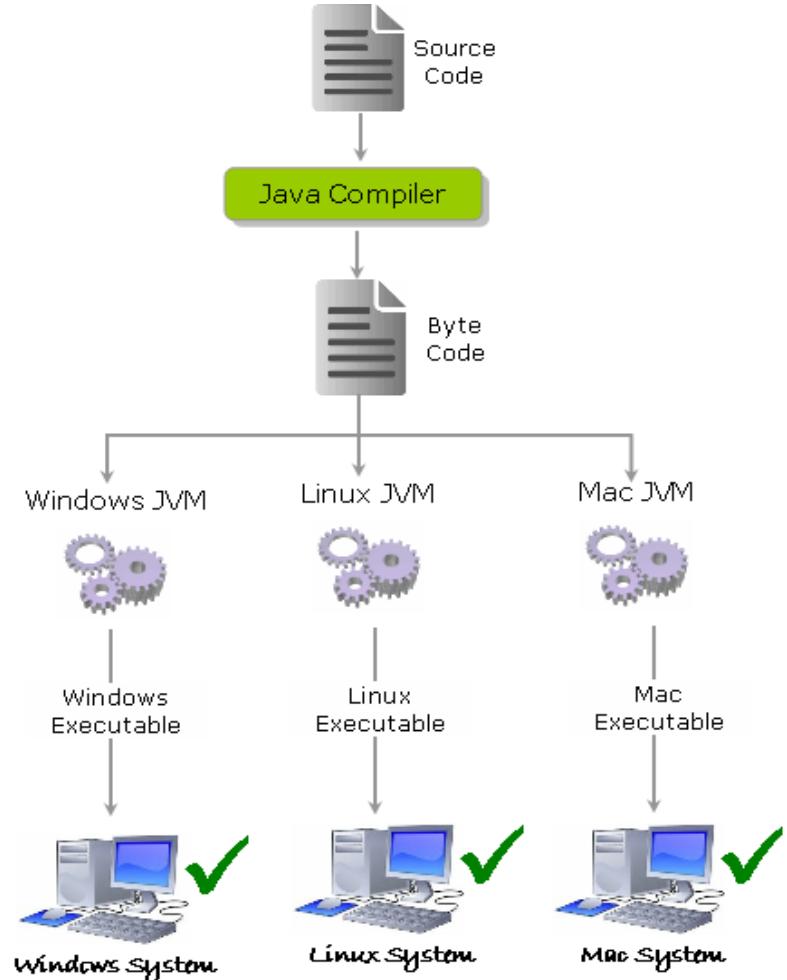


(a)



Virtuali mašina (JVM)

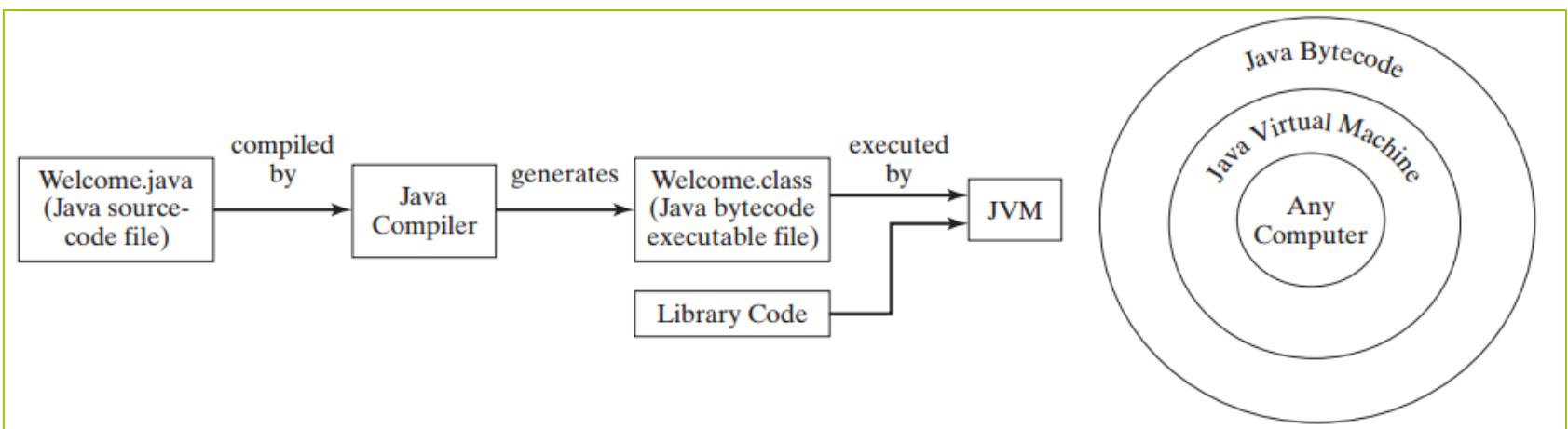
- Java Virtual Machine
- Vykdo Java kodą
- Optimizuoja esamai sistemai
- Abstrakcija užtikrinanti programų nepriklausomumą nuo platformos

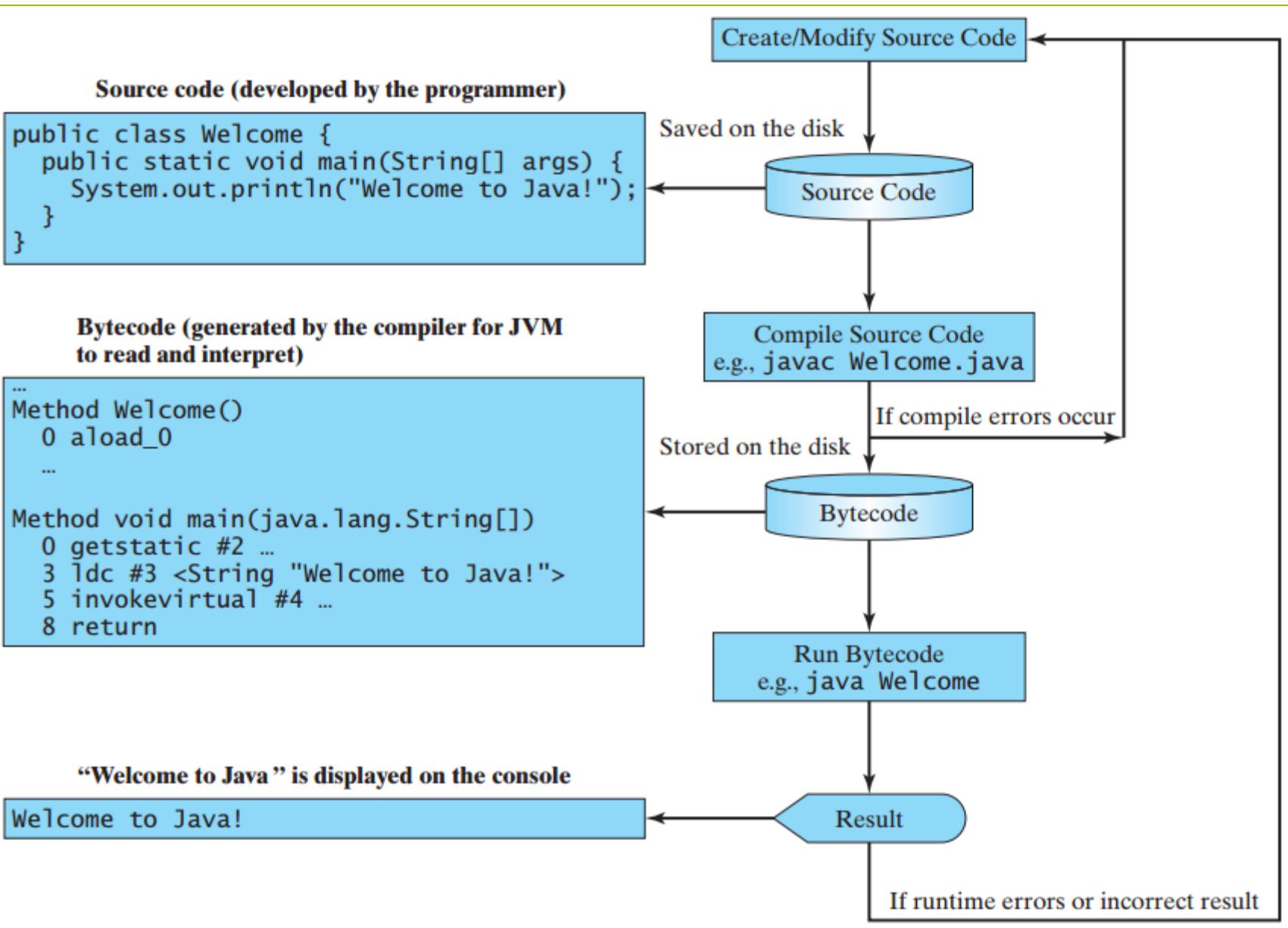


Write Once Run Anywhere

Bytecode

- Java programa kompiliuojant paverčiama į JVM komandas, o ne konkretaus kompiuterio procesoriaus komandas.
- Sukompiliuota Java programa (**bytecode**) vienodai veiks ant visų kompiuterių, kur yra įdiegta JVM
- JVM atlieka baitinio kodo interpretavimą (vykdymą)





JRE

- **Java runtime environment (JRE)** sudaro Java Virtuali Mašina(JVM), java platformos baziņes klasēs ir pagalbinēs Java platformos bibliotekos.
- JRE ļeina į naršykles, operacines sistemas, duomenų bazių sistemas ir pan.
- Pagrindinė JRE komanda yra *java*, kuri atlieka baitinio kodo vykdymā (interpretavimā):

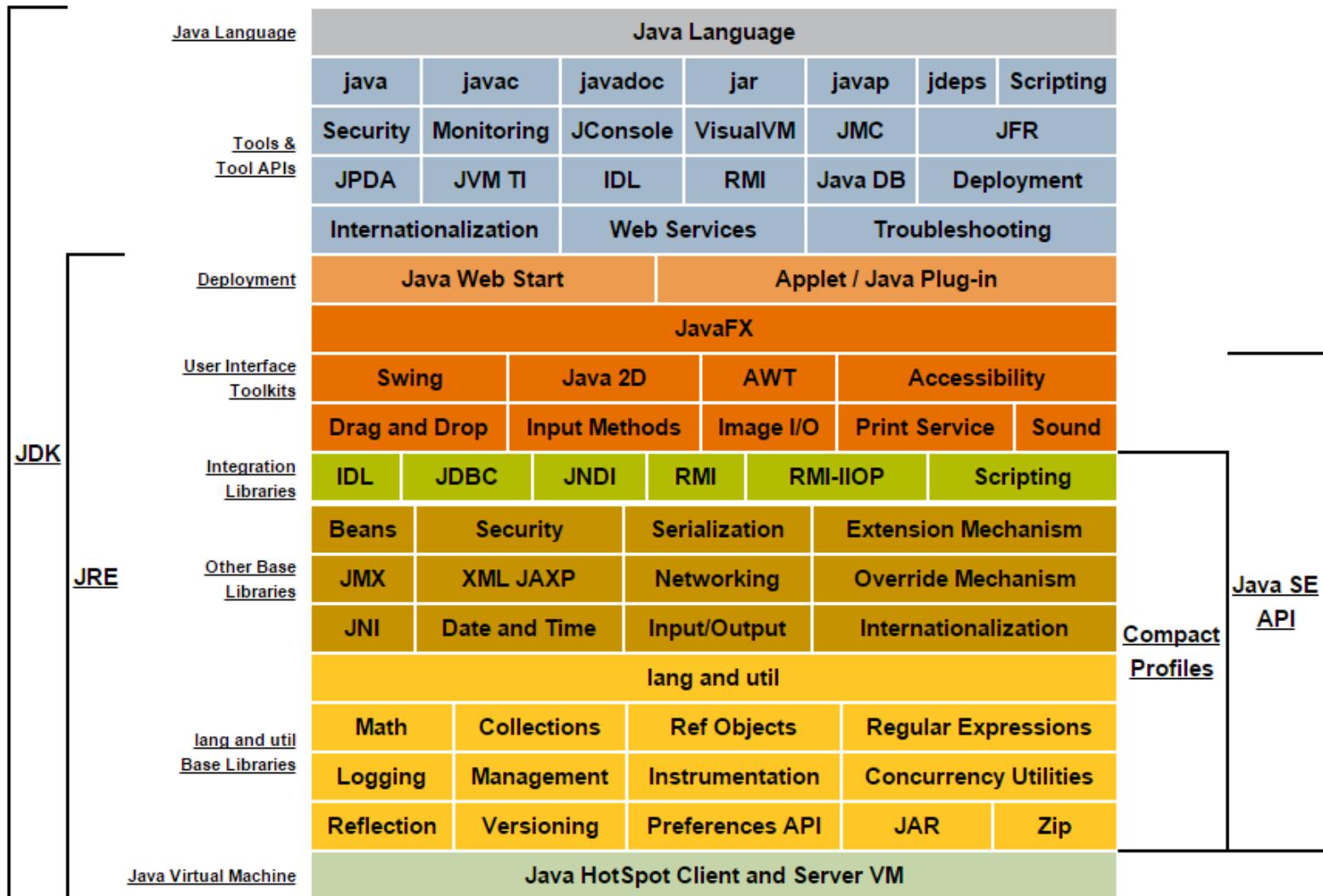
```
C:\javaworks\helloworld>java helloworld  
Hello World!
```

JDK

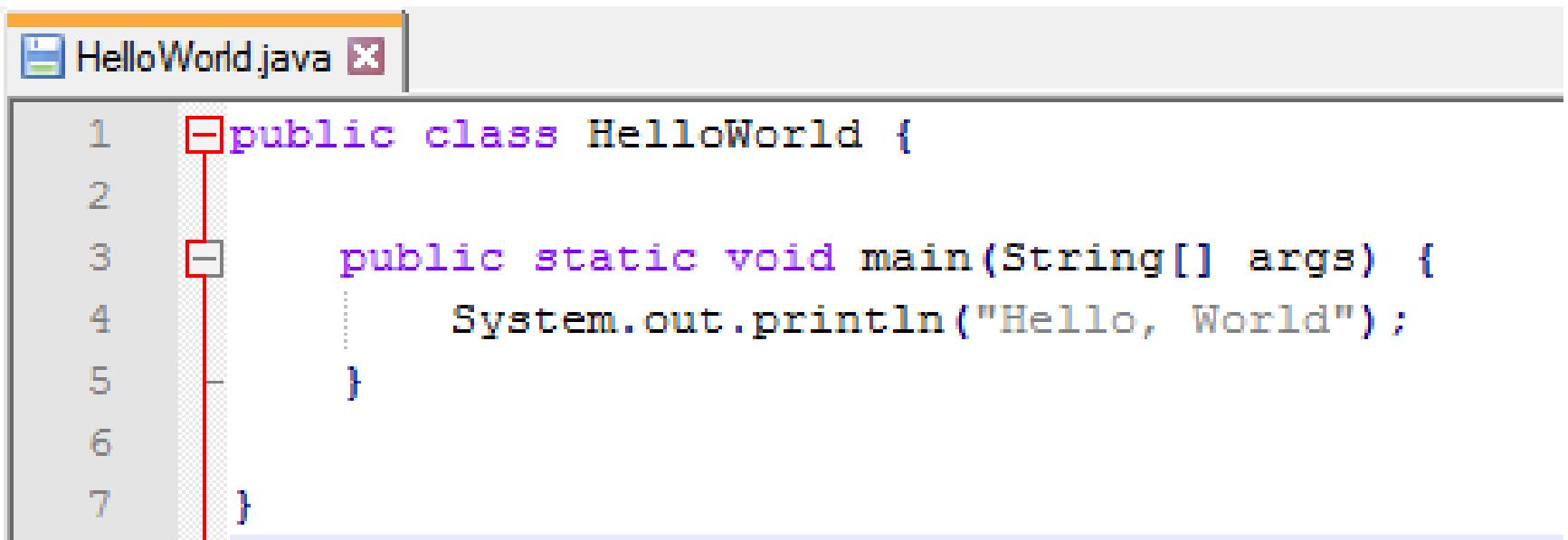
- Norint sukompiliuoti ir vykdyti Java kodą, reikia įsidiegti **Java Development Kit (JDK)**.
- Pagrindinė JDK komanda yra ***javac***, kuri kompiliuoja Java kodą

```
C:\javaworks\helloworld>javac helloworld.java
```

JRE yra JDK poaibis



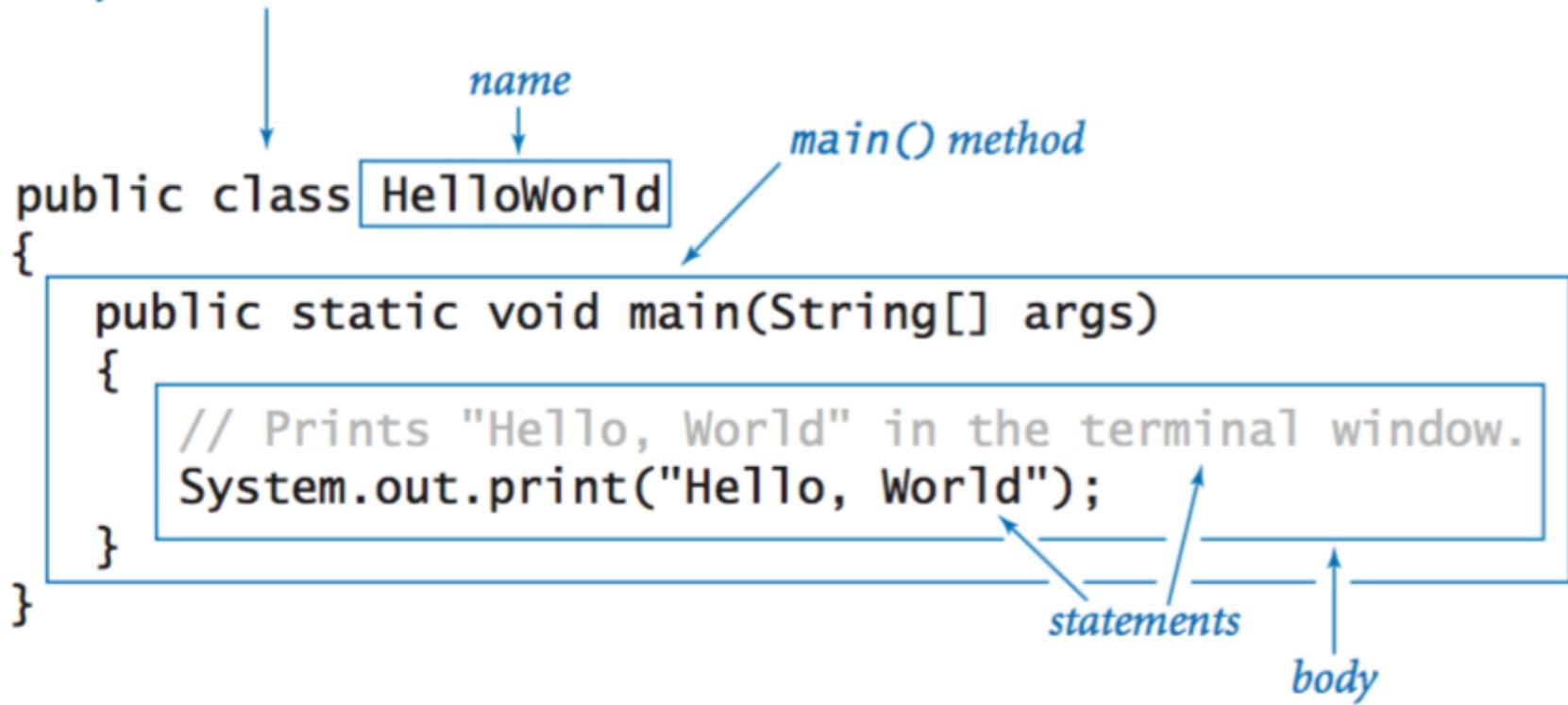
Kaip atrodo Java kalba?



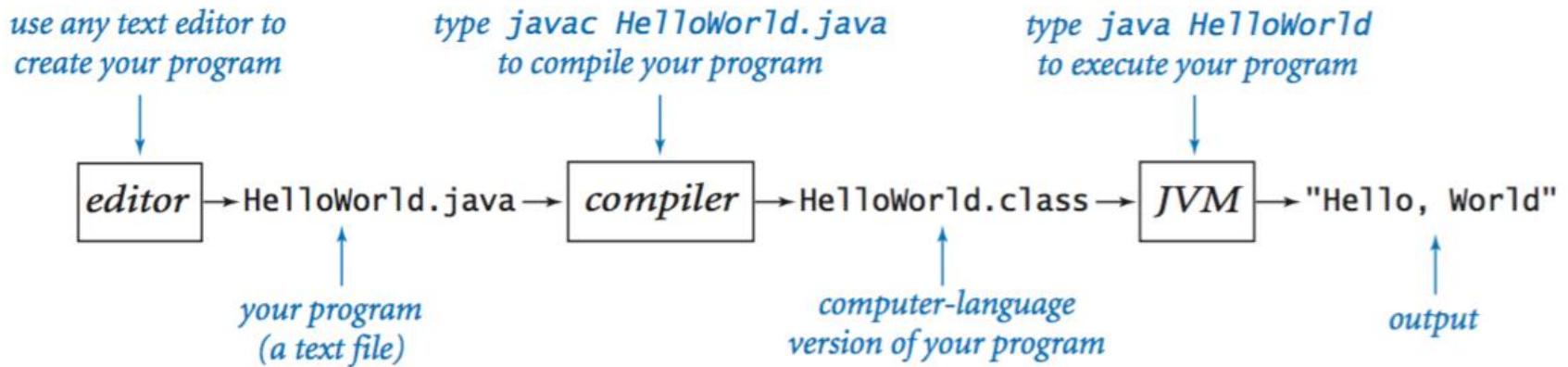
```
1 public class HelloWorld {  
2  
3     public static void main(String[] args) {  
4         System.out.println("Hello, World");  
5     }  
6  
7 }
```

Java HelloWorld

text file named `HelloWorld.java`



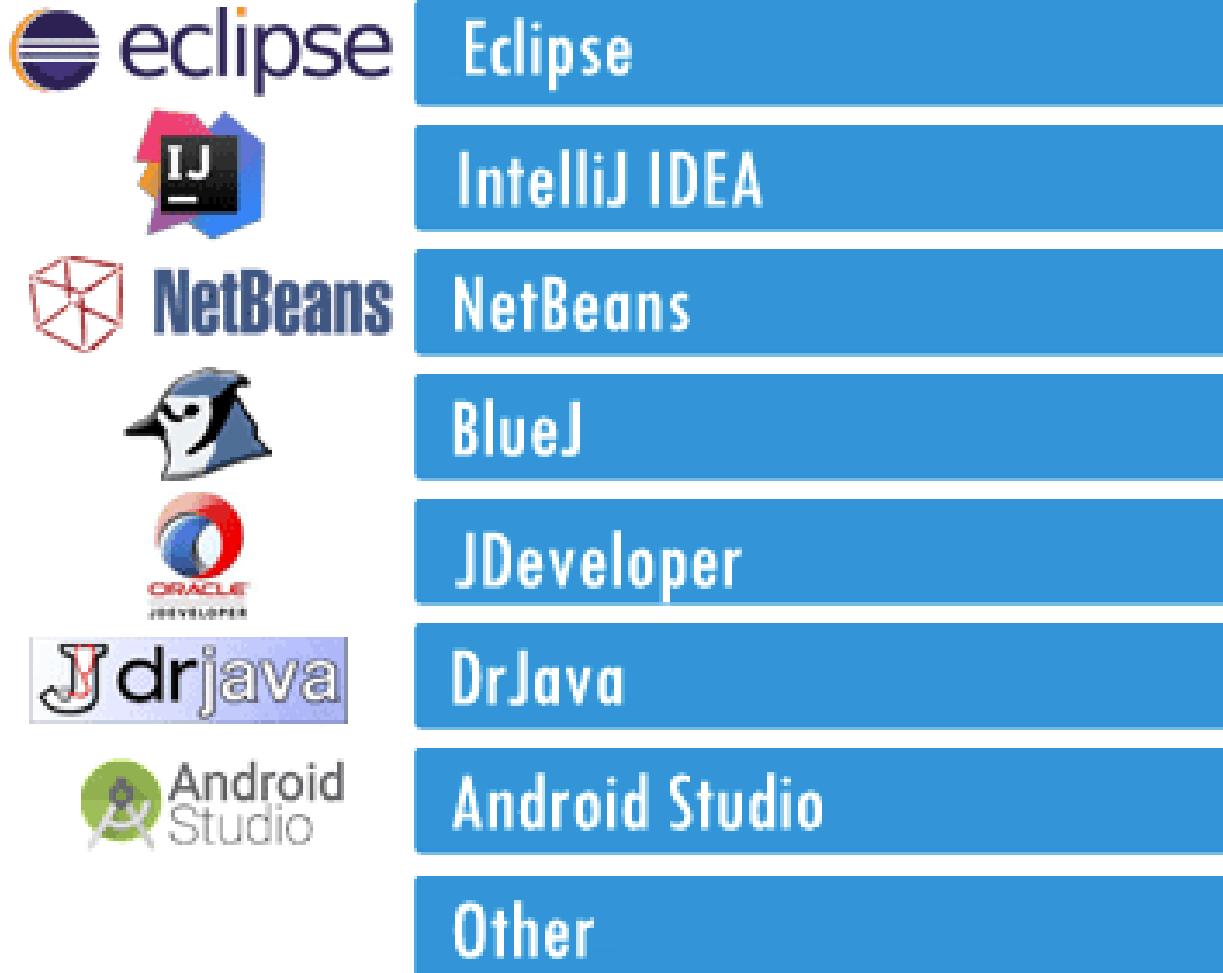
Kompiliavimas ir paleidimas



IDE (Integruotos kūrimo aplinkos)

- Integrated Development Environment
- Palengvina kūrimo procesą
 - Palengvina darbą su kodu (užuominos, šablonai, informacija apie klaidas...)
 - Palengvina darbą su dideliais projektais
 - Integruoja daug įrankių į vieną paketą (kompiliatorių, surinkėją, derintoją, versijų valdymo sistemą, kodo kokybės inspektavimo įrankius, ...)

IDE (Integruotos kūrimo aplinkos)



Praktinis darbas

- Įdiegti JDK (nepamirškit pakeisti PATH Environment variable)
- Patikrinti versijas (*java -version, javac -version*)
- Teksto redaktoriuje parašyti „HelloWorld“ programą ir paleisti iš komandinės eilutės (*javac, java*)
- Įdiegti Eclipse IDE for Java EE Developers (nurodykit workspace vietą)
- * *Įdiegti IntelliJ IDEA Community edition*



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

02 – JAVA PAGRINDAI

Jaroslav Grablevski / Justina Balsė

Turinys

- Pirmoji programa;
- Išvedimo sakinys;
- Komentarai;
- Kintamieji*;
- Aritmetiniai operatoriai*;
- Duomenų įvedimas;
- Matematikos klasė *Math*;

*Tik pagrindai. Plačiau nagrinėsime vėliau.

Hello World | Java

```
public class PirmojiPrograma {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

```
public class PirmojiPrograma {  
  
    public static void main(String[] args) {  
  
        // Sentence 1  
        // Sentence 2  
        // ...  
        // Sentence n  
  
    }  
}
```

Išvedimo sakinys (1)

```
public class PirmojiPrograma {  
  
    public static void main(String[] args) {  
  
        System.out.println("Hello World!");  
        System.out.println("Labas pasauli!");  
        System.out.println("...123@#$%^");  
  
    }  
}
```

Hello World!
Labas pasauli!
...123@#\$%^

Išvedimo sakinys (2)

```
public class PirmojiPrograma {  
  
    public static void main(String[] args) {  
  
        System.out.println("Sakinys 1"); System.out.println("Sakinys 2");  
  
        System.out.println("=====");  
  
        System.out.print("Sakinys 3"); System.out.print("Sakinys 4");  
    }  
}
```

Sakinys 1
Sakinys 2
=====

Sakinys 3Sakinys 4

Kiekviena
komanda/sakinys
atskiriamas
kabliataškiu.

System.out.println(); | System.out.print();

Komentarai | Comments (1)

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

Komentarai | Comments (2)

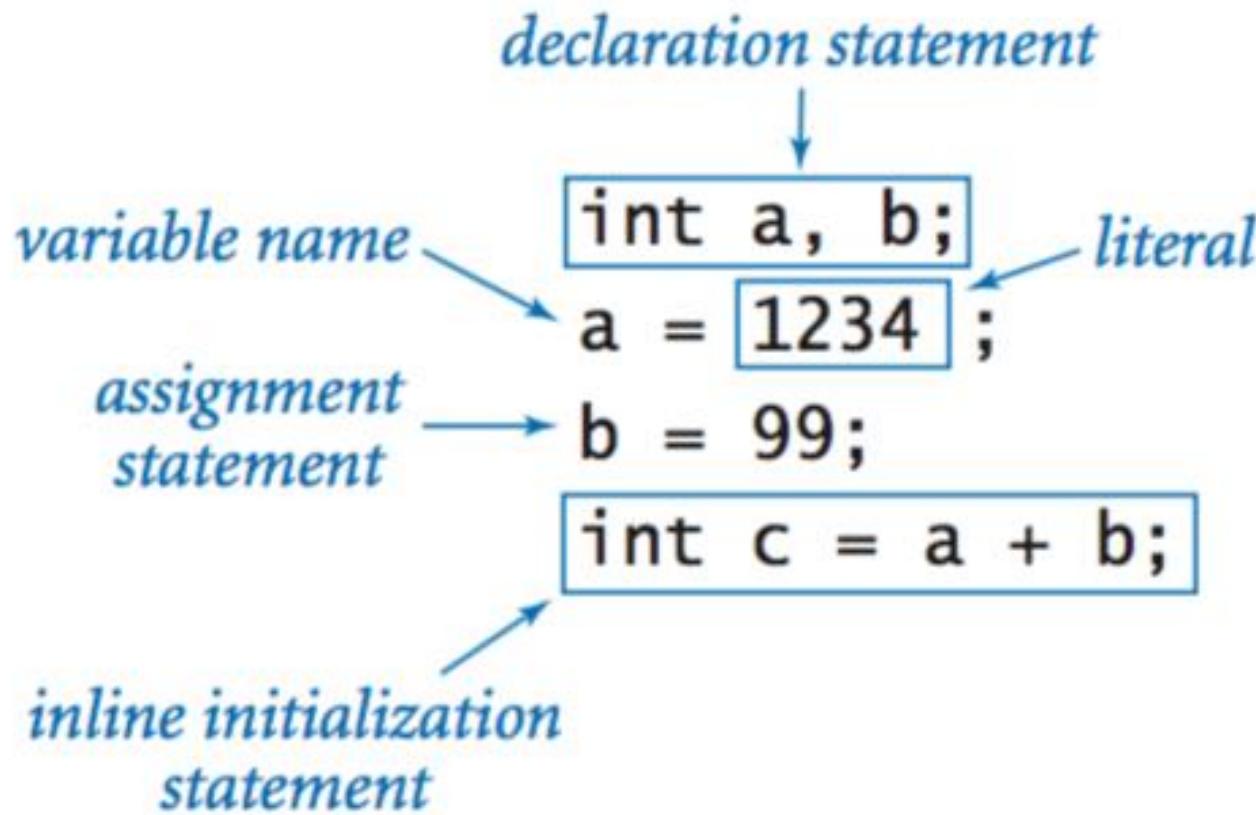
```
public class PirmojiPrograma {  
  
    public static void main(String[] args) {  
  
        // this is a single-line comment  
        System.out.print("...");  
  
        System.out.print("..."); // a single-line comment after code  
  
        /* This is also a  
         * comment spanning  
         * multiple lines */  
  
    }  
}
```

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



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 $\pm 4.94065645841246544 \times 10^{-324}$

Kintamuju 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

Kintamieji | Variables

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

Kintamuju 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
```

Kintamuju 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!

Kintamuju išvedimas (3)

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

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

x = 5, y = -9

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
```

Duomenų įvedimas (1)

```
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 = reader.nextInt();

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

        reader.close();

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

Duomenų įvedimas (2)

```
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 elementarijas matematines operacijas.
- Turi dvi konstantas: eulerio (E) ir pi (PI);

Matematikos klasė *Math* (2)

```
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);
```

02 - Praktiniai darbai

1. 02 - NPIK - Java pagrindai (PraktikaEN);
2. 02 - NPIK - Java pagrindai (PraktikaLT);



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

03 - ŠAKOTIEJI ALGORITMAI

Jaroslav Grablevski / Justina Balsė

Turinys

- Lyginimo operatoriai;
- *boolean* tipas;
- Šakotieji algoritmai (`if-else` ir `switch`);
- AND
- OR
- Didžiausios/mažiausios reikšmės paieškos algoritmas

Lyginimo operatoriai | Relational Operators

```
(a == b)    // (ar lygu)
(a != b)    // (ar nelygu)
(a > b)     // (a daugiau už b)
(a >= b)    // (a daugiau arba lygu b)
(a < b)     // (a mažiau už b)
(a <= b)    // (a mažiau arba lygu už b)
```

boolean tipo kintamasis

```
int first = 1;  
int second = 3;  
  
boolean isGreater = first > second;  
  
System.out.println(isGreater);
```

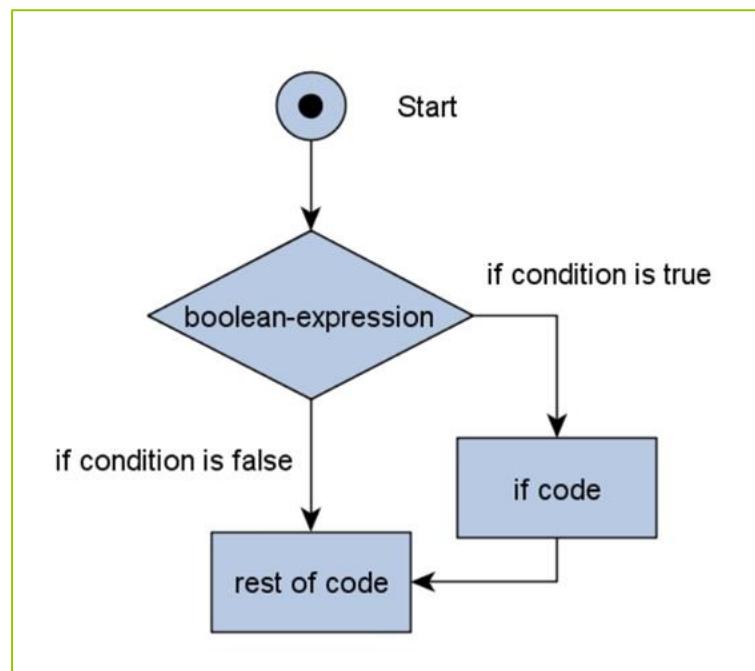
TRUE

FALSE

Šakotieji algoritmai (if)

```
// if-then  
if ( booleanExpression ) {  
    true-block ;  
}
```

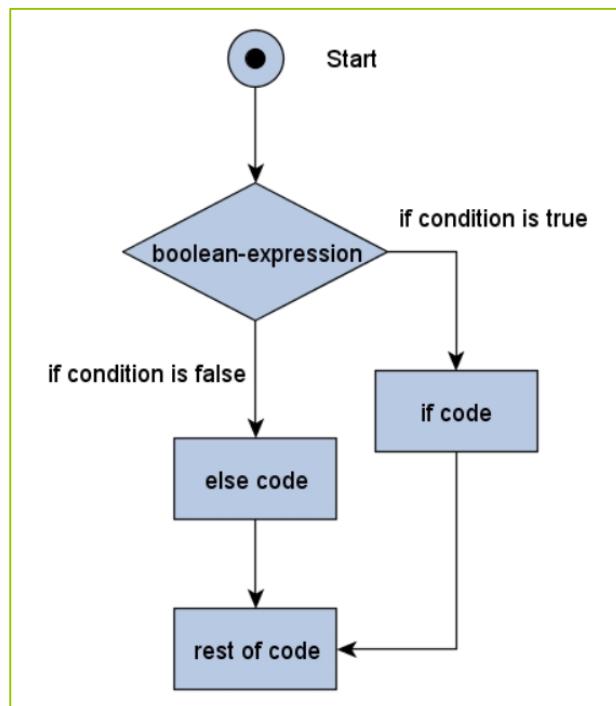
```
if (mark >= 50) {  
    System.out.println("Congratulation!");  
    System.out.println("Keep it up!");  
}
```



Šakotieji algoritmai (if-else)

```
// if-then-else
if ( booleanExpression ) {
    true-block ;
} else {
    false-block ;
}
```

```
if (mark >= 50) {
    System.out.println("Congratulation!");
    System.out.println("Keep it up!");
} else {
    System.out.println("Try Harder!");
}
```

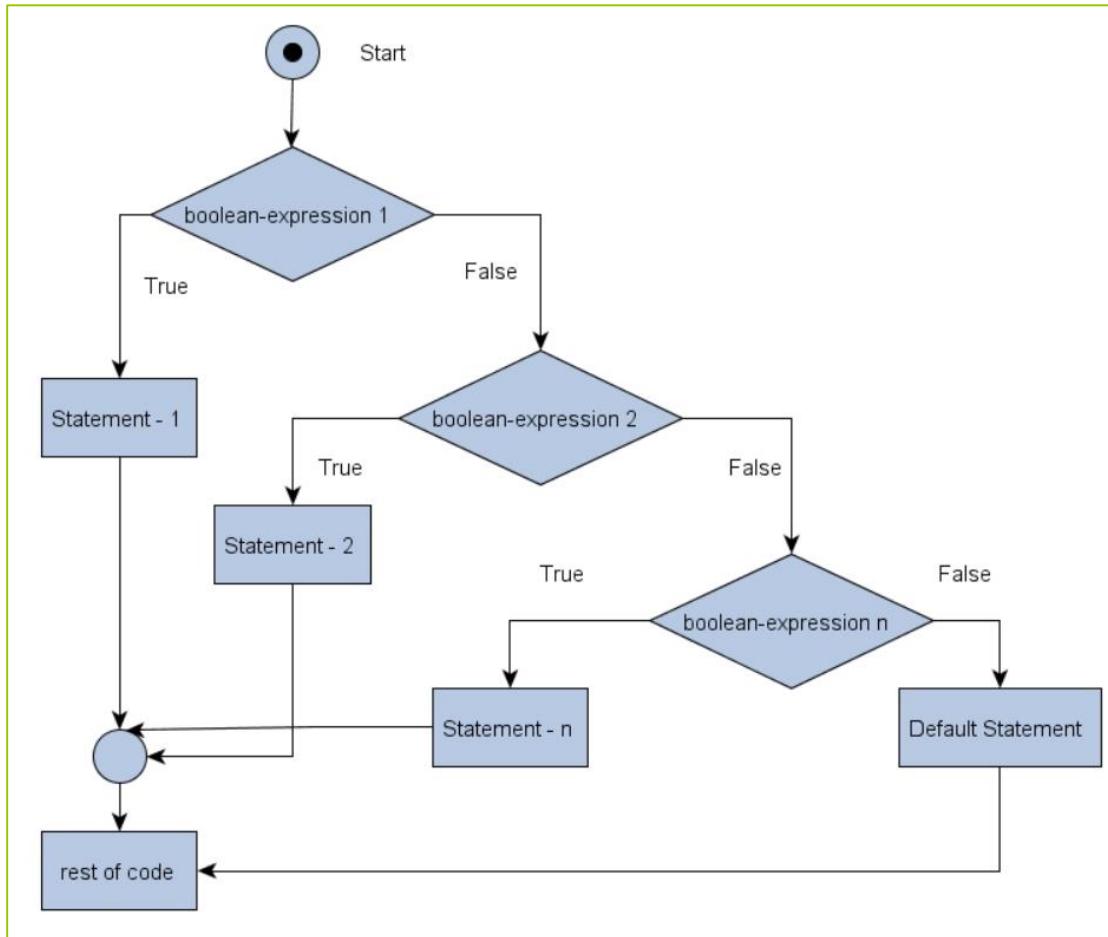


Šakotieji algoritmai (if-else ladder)

```
// nested-if
if ( booleanExpr-1 ) {
    block-1 ;
} else if ( booleanExpr-2 ) {
    block-2 ;
} else if ( booleanExpr-3 ) {
    block-3 ;
} else if ( booleanExpr-4 ) {
    .....
} else {
    elseBlock ;
}
```

```
if (mark >= 80) {
    System.out.println("A");
} else if (mark >= 70) {
    System.out.println("B");
} else if (mark >= 60) {
    System.out.println("C");
} else if (mark >= 50) {
    System.out.println("D");
} else {
    System.out.println("F");
}
```

Šakotieji algoritmai (if-else ladder)



Pavyzdys (1)

```
int number = 11;  
  
if (number > 10) System.out.println("The number was greater than 10");
```

```
int number = 11;  
  
if (number > 10) {  
    System.out.println("The number was greater than 10");  
}
```

Pavyzdys (2)

```
int number = 4;

if (number > 5) {
    System.out.println("Your number is greater than five!");
} else {
    System.out.println("Your number is equal to or less than five!");
}
```

```
int number = 4;

if (number > 5) {
    System.out.println("Your number is greater than five!");
    System.out.println("... ");
    System.out.println("... ");
} else {
    System.out.println("Your number is equal to or less than five!");
    System.out.println("... ");
    System.out.println("... ");
}
```

Pavyzdys (3)

```
int number = 3;

if (number == 1) {
    System.out.println("The number is one.");
} else if (number == 2) {
    System.out.println("The number is two.");
} else if (number == 3) {
    System.out.println("The number is three!");
} else {
    System.out.println("Quite a lot!");
}
```

AND &&

```
System.out.println("Is the number between 5-10?");
int number = 7;

if (number > 4 && number < 11) {
    System.out.println("Yes! :)");
} else {
    System.out.println("Nope :( ");
}
```

(*sąlyga1* && *sąlyga2*)
Jeigu abi sąlygos yra teisingos,
tai visa (bendra) sąlyga yra teisinga

Didžiausios/mažiausios reikšmės paieška

```
int a = 10, b = 15, c = 5;
int max;

if((a > b) && (a > c)){
    max = a;
}else if ((b > c) && (b > a)){
    max = b;
}else
    max = c;
System.out.println("Max: " + max);
```

OR ||

```
System.out.println("Is the number less than 0 or greater than 100?");
int number = 145;

if (number < 0 || number > 100) {
    System.out.println("Yes! :)");
} else {
    System.out.println("Nope :( ");
}
```

(*sąlyga1* || *sąlyga2*)
Jeigu **nors viena sąlyga teisinga**,
tai visa (bendra) sąlyga yra teisinga

LOGICAL OPERATORS

Operators && (AND)	Operator (OR)	Operator ! (NOT)
true && true → true	true true → true	!true → false
true && false → false	true false → true	!false → true
false && true → false	false true → true	
false && false → false	false false → false	
true && true && false → false	false false true → true	

String tipo kintamuju palyginimas

```
String text = "course";

if (text.equals("marzipan")) {
    System.out.println("The variable text contains the text marzipan");
} else {
    System.out.println("The variable text does not contain the text marzipan");
}
```

```
String text = "course";
String anotherText = "horse";

if (text.equals(anotherText)) {
    System.out.println("The texts are the same!");
} else {
    System.out.println("The texts are not the same!");
}
```

Šakotieji algoritmai (2)

```
// switch-case-default
switch ( selector ) {
    case value-1:
        block-1; break;
    case value-2:
        block-2; break;
    case value-3:
        block-3; break;
    .....
    case value-n:
        block-n; break;
    default:
        default-block;
}
```

```
char oper; int num1, num2, result;
.....
switch (oper) {
    case '+':
        result = num1 + num2; break;
    case '-':
        result = num1 - num2; break;
    case '*':
        result = num1 * num2; break;
    case '/':
        result = num1 / num2; break;
    default:
        System.out.println("Unknown operator");
}
```

Pavyzdys (4)

```
char grade = 'C';

switch(grade) {
    case 'A' :
        System.out.println("Excellent!");
        break;
    case 'B' :
    case 'C' :
        System.out.println("Well done");
        break;
    case 'D' :
        System.out.println("You passed");
    case 'F' :
        System.out.println("Better try again");
        break;
    default :
        System.out.println("Invalid grade");
}
System.out.println("Your grade is " + grade);
```

03 – Praktiniai darbai

- 03 - NPIK - Sakotieji algoritmai (PraktikaEN)
- 03 - NPIK - Sakotieji algoritmai (PraktikaLT)



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

4 – CIKLINIAI ALGORITMAI

Jaroslav Grablevski / Justina Balsė

Turinys

- Increment/Decrement;
- Nežinomo kartojimų skaičiaus ciklas WHILE;
- *break*
- Žinomo kartojimų skaičiaus ciklas FOR;
- Sumos algoritmas
- Kiekio algoritmas
- *continue*

Increment/Decrement

```
int x = 5;

// Increment
x++;      // x = x + 1;
System.out.println("x = " + x); // 6

// Decrement
int y = 10;
y--;      // y = y - 1;
System.out.println("y = " + y); // 9
```

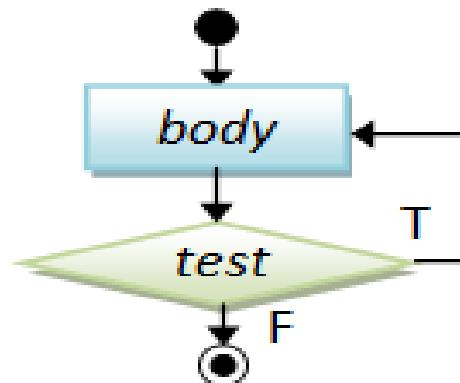
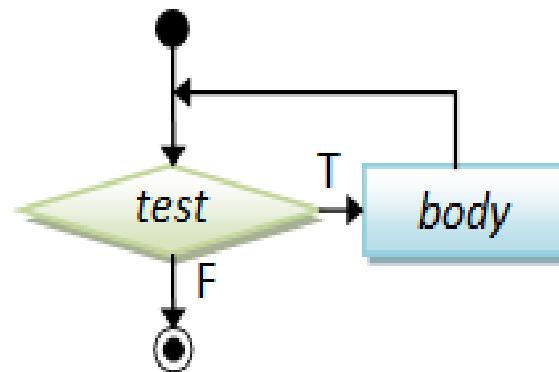
Prefix/postfix

Operator	Operator name	Sample expression	Explanation
<code>++</code>	prefix increment	<code>++a</code>	Increment <code>a</code> by 1, then use the new value of <code>a</code> in the expression in which <code>a</code> resides.
<code>++</code>	postfix increment	<code>a++</code>	Use the current value of <code>a</code> in the expression in which <code>a</code> resides, then increment <code>a</code> by 1.
<code>--</code>	prefix decrement	<code>--b</code>	Decrement <code>b</code> by 1, then use the new value of <code>b</code> in the expression in which <code>b</code> resides.
<code>--</code>	postfix decrement	<code>b--</code>	Use the current value of <code>b</code> in the expression in which <code>b</code> resides, then decrement <code>b</code> by 1.

Nežinomo kartojimų skaičiaus ciklas WHILE

```
// while-do loop  
while ( test ) {  
    body;  
}
```

```
// do-while loop  
do {  
    body;  
}  
while ( test );
```



Pavyzdys while

```
int i = 0;  
while (i < 11) {  
    System.out.print(i + " ");  
    i++;  
}  
// 0 1 2 3 4 5 6 7 8 9 10
```

Pavyzdys do while

```
int count = 1;  
do {  
    System.out.print(count + " ");  
    count++;  
} while (count < 11);
```

1 2 3 4 5 6 7 8 9 10

Pavyzdys infinite loop

```
while (true) {  
    System.out.println("I can program!");  
}
```

```
int i=10;  
while(i>1)  
{  
    System.out.println(i);  
    i++;  
}
```

Pavyzdys | break

```
Scanner reader = new Scanner(System.in);

while (true) {
    System.out.println("I can program!");

    System.out.print("Continue? ('no' to quit)? ");
    String command = reader.nextLine();
    if (command.equals("no")) {
        break;
    }
}

System.out.println("Thank you and see you later!");

reader.close();
```

```
I can program!
Continue? ('no' to quit)? Hi
I can program!
Continue? ('no' to quit)? Hello
I can program!
Continue? ('no' to quit)? Bye
I can program!
Continue? ('no' to quit)? no
Thank you and see you later!
```

Praktika

04 – Praktiniai darbai

04 - Cikliniai algoritmai (PraktikaEN)

Žinomo kartojimų skaičiaus ciklas FOR

```
for /*Initialization*/ ; /*Condition*/ ; /* Iteration */ {  
    /* loop body */  
}
```

```
for /*inicjalizacija*/; /*loginė-išraiška*/; /*kitimo-žingsnis*/ {  
    // kartojami sakiniai  
}
```

Pavyzdys (1)

```
for (int i = 0; i<6; i++) {  
    System.out.println("i is " + i);  
}
```

```
i is 0  
i is 1  
i is 2  
i is 3  
i is 4  
i is 5
```

Pavyzdys (2)

```
public static void main(String[] args) {  
  
    int a = 5, b = 9;  
  
    for(int i=a; i<b; i++){  
        System.out.println("i = " + i);  
    }  
}
```

```
i = 5  
i = 6  
i = 7  
i = 8
```

```
public static void main(String[] args) {  
  
    int a = 5, b = 15;  
  
    for(int i=a; i<b; i=i+2){  
        System.out.println("i = " + i);  
    }  
}
```

```
i = 5  
i = 7  
i = 9  
i = 11  
i = 13
```

Pavyzdys (3)

```
int a = 1;
for(int i=5; i<11; i++){
    a = a * i;
    System.out.println("Kai i = " + i + ", tai a = " + a);
}
```

```
Kai i = 5, tai a = 5
Kai i = 6, tai a = 30
Kai i = 7, tai a = 210
Kai i = 8, tai a = 1680
Kai i = 9, tai a = 15120
Kai i = 10, tai a = 151200
```

Pavyzdys (4) | FOR ir IF

Parenkite programą, kuri atspausdintų visus dviženklius skaičius dalius iš 6.

```
for (int i=10; i<100; i++){
    if (i % 6 == 0)
        System.out.print(i + " ");
}
```

12	18	24	30	36	42	48	54	60	66	72	78	84	90	96
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Pavyzdžiai* (5)

```
for (int x = 1; x < 2; x++) {  
    System.out.println(x); // Legal  
}  
System.out.println(x); // Not Legal! x is now out of scope  
// and can't be accessed.
```

```
for (int i = 0, j = 0; (i < 10) && (j < 10); i++, j++) {  
    System.out.println("i is " + i + " j is " + j);  
}
```

```
int i = 0;  
for (; i < 10;) {  
    i++;  
    // do some other work  
}
```

Sumos algoritmas

```
int sum = 0;  
for (int i = 0; i < 11; i++) {  
    sum = sum + i; // sum+=i;  
}  
System.out.println("sum = " + sum);
```

sum = 55

Kiekio algoritmas

```
int count = 0;
for (int i = 13; i < 24; i++) {
    if (i % 3 == 0) {
        count = count + 1; // count++
    }
}
System.out.println("count = " + count); // count = 3
```

Nested loops

```
int size = 8;
for (int row = 1; row <= size; ++row) { // Outer loop
    for (int col = 1; col <= size; ++col) { // Inner loop
        System.out.print("# ");
    }
    System.out.println();
}
```

A 9x9 grid of black '#' symbols, arranged in a single continuous line.

continue

```
double f = 0;
for(int i=-3; i<4; i++){
    if (i == 0)
        continue;
    f = (double) 1 / i;
    System.out.println("f(" + i + ") = " + f);
}
```

```
f(-3) = -0.3333333333333333
f(-2) = -0.5
f(-1) = -1.0
f(1) = 1.0
f(2) = 0.5
f(3) = 0.3333333333333333
```

labeled *continue* and *break*

```
boolean.isTrue = true;
outer: for (int i = 0; i < 5; i++) {
    while (isTrue) {
        System.out.println("Hello");
        break outer;
    } // end of inner while loop
    System.out.println("Outer loop."); // Won't print
} // end of outer for loop
System.out.println("Good-Bye");
```

Hello
Good-Bye

Praktika

04 – Praktiniai darbai:

04 - Cikliniai algoritmai (PraktikaLT)

04 - Cikliniai algoritmai (PraktikaNestedLoops)



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

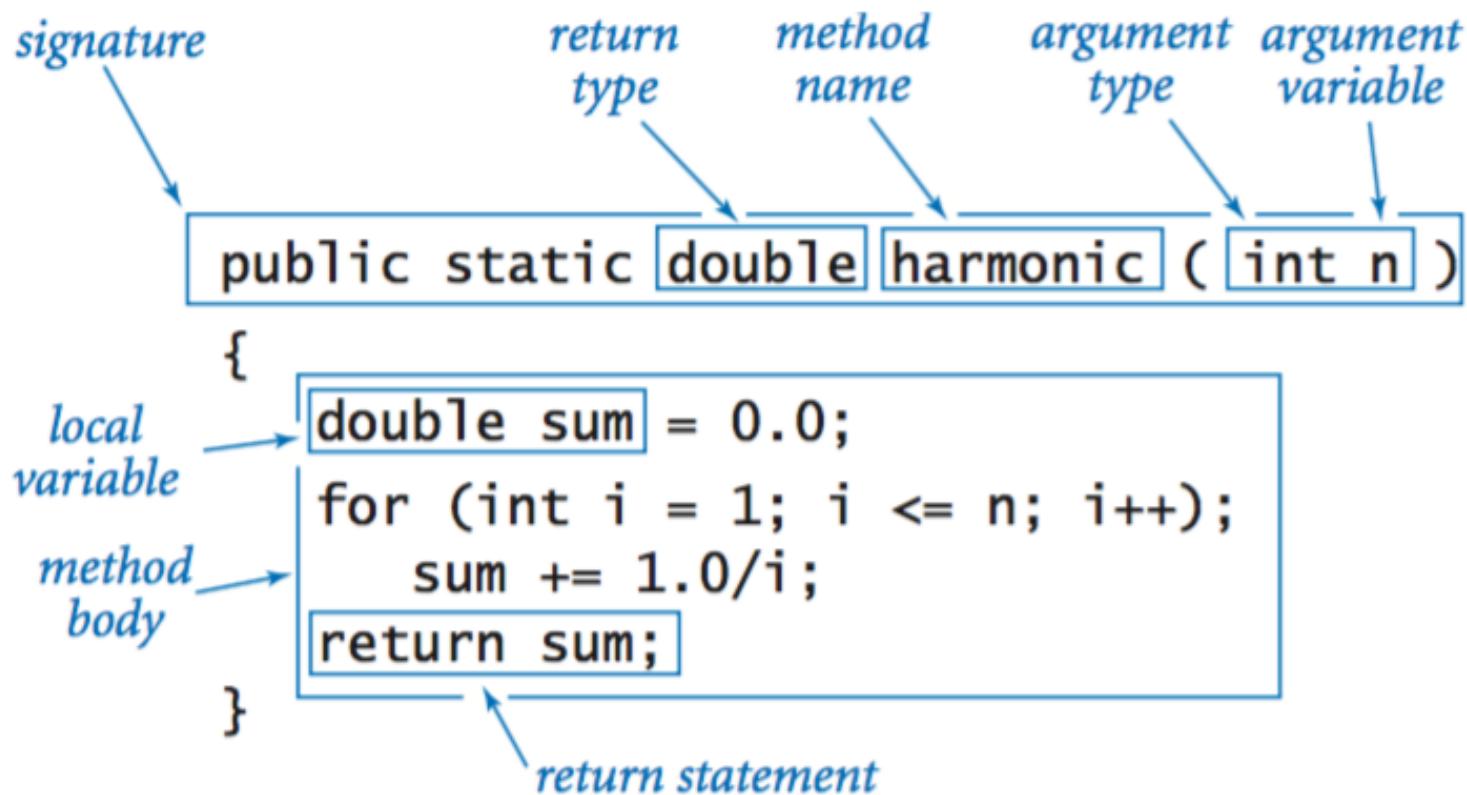
4 - METODAI

Jaroslav Grablevski / Justina Balsė

Turinys

- Metodai grąžinantys reikšmę
- Metodai negrąžinantys reikšmės
- Uždavinių sprendimo pavyzdžiai

Metodai



Method modifiers (modifikatoriai)*

Access control modifiers (matomumo modifikatoriai):

public	matomumo sritis neribojama (jei matoma jo klasė)
<i>(pagal nutylėjimą)</i>	matomi savo paketo klasėse
protected	matomi savo paketo ir išvestinėse klasėse
private	matomi savo klasėje

static	galima taikyti klasei (be objekto)
abstract	metodas neturi kūno
final	metodas negali būti užklotas
native	parašytas ne javos kalba
synchronized	tuo pačiu metu metodą gali naudoti tik vienas procesas (thread)
strictfp	apribojimai skaičiavimams su slankiuoju kableliu

*plačiau bus kai nagrinėsime objektinį programavimą

Rezultato tipas (return type)

- Metodo antraštėje turime nurodyti vieną rezultato tipą (pvz. double, boolean, String, int[], Person)
- Metodo apraše (kūne) grąžinamą rezultatą nurodome raktiniu žodžiu return
- Jei metodas nieko negražina, turime nurodyti atributą void

Parametru sąrašas (argument list)

- Parametru sąrašas sudarytas iš kintamųjų tipų ir vardų poru, atskirtų kableliais. Pvz: (String name, int age)
- Sąrašas gali būti tuščias: ()
- Metodo parametrai galioja tik tam metodui

Metodai

```
public static void main(String[] args) {  
    greet();  
    greet();  
    greet();  
}  
  
public static void greet() {  
    System.out.println("Greetings from the world of methods!");  
}
```

Greetings from the world of methods!
Greetings from the world of methods!
Greetings from the world of methods!

Metodai su parametrais

```
public static void main(String[] args) {  
    greet("Jonas");  
    greet("Justina");  
}  
  
public static void greet(String name) {  
    System.out.println("Hi " + name + ", greetings from the world of methods!");  
}
```

Hi Jonas, greetings from the world of methods!
Hi Justina, greetings from the world of methods!

Metodai – negražinantys reikšmės

```
public class Metodai {  
  
    public static void main(String[] args) {  
  
        // Raskite stac plota  
        int a = 5, b = 6;  
        raskPlot(a, b);  
  
    }  
    public static void raskPlot(int a, int b){  
  
        int S = a * b;  
        System.out.println("S = " + S);  
    }  
}
```

Metodai – negražinantys reikšmės

```
public static void simpleMethod() {  
    if (Math.random() > 0.5) {  
        System.out.println("Didesnis skaičius");  
        return; // explicit return  
    } else {  
        System.out.println("Mažesnis skaičius");  
    }  
    System.out.println("Pabaiga");  
    //rezultato tipas yra "void", todėl galime  
    //nerašyti raktinio žodžio "return"  
}
```

Metodai – gražinantys reikšmę

```
public class Metodai {  
  
    public static void main(String[] args) {  
  
        // Raskite stac plota  
        int a = 5, b = 6;  
        int s;  
  
        s = raskPlota(a, b);  
  
        System.out.println("S = " + s);  
        //System.out.println("S = " + raskPlota(a,b));  
    }  
  
    public static int raskPlota(int a, int b){  
  
        int plotas;  
        plotas = a * b;  
        return plotas;  
        //return a*b;  
    }  
}
```

Metodai – gražinantys reikšmę

```
public static String returningMethod() {  
    if(Math.random()>0.5) {  
        return "Išėjom ankščiau";  
    }  
    //return 100; - netinka, nes rezultato tipas kitoks  
    return "Metodo pabaiga";  
}  
  
//return a*b;
```

Pavyzdys (1)

- Duotas sveikujų skaičių intervalas $[m, n]$.
- Parašykite programą, kuri suskaičiuotų kiek tame intervale yra skaičių, kurie dalinasi iš juos sudarančių skaitmenų sumos.
- **Metodas** – grąžinantis vieno skaičiaus skaitmenų sumą.

Sprendimas (1) Pagrindinis metodas main

```
public static void main(String[] args) {  
  
    Scanner rd = new Scanner(System.in);  
  
    int kiek = 0;  
  
    // duomenu ivedimas  
    System.out.print("Iveskite m: ");  
    int m = rd.nextInt();  
    System.out.print("Iveskite n: ");  
    int n = rd.nextInt();  
  
    for(int i=m; i<=n; i++){  
        if(i % skaitmenuSuma(i) == 0)  
            kiek++;  
    }  
  
    System.out.println("Skaiciu kiekis: " + kiek);  
}
```

Sprendimas (2) Metodas **skaitmenuSuma**

```
public static int skaitmenuSuma(int sk){  
    int suma = 0;  
    int psk;  
    while(sk > 0){  
        psk = sk % 10;  
        suma = suma + psk;  
        sk = sk / 10;  
    }  
    return suma;  
}
```

Variable length argument list (varArgs)

- Trys taškai už parametru tipo reišia, kad argumentų skaičius gali būti bet koks.
- Metode šie argumentai naudojami kaip masyvo elementai.

```
public static int addUpNumbers(int... values) {
    int sum = 0;
    for (int v: values) {
        sum +=v;
    }
    return sum;
}

public static void main(String[] args) {
    System.out.println(addUpNumbers(1,2,3));
    System.out.println(addUpNumbers(1,2,3,4,5,6,7));
    System.out.println(addUpNumbers());
    System.out.println(addUpNumbers(new int[]{1,2,3})); //masyvas
}
```

Pavyzdys

```
public static boolean isEven(int sk){  
    boolean is = false;  
  
    if (sk % 2 == 0)  
        is = true;  
    else  
        is = false;  
  
    return is;  
}
```

```
public static boolean isEven(int sk) {  
    return sk % 2 == 0;  
}
```



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

6- MASYVAI

Jaroslav Grablevski / Justina Balsė

Masyvai

- Masyvai skirti saugoti dideliam kiekiui vienodo tipo reikšmių
- Masyvo elementai skaičiuojami nuo 0 (žymi pirmą elementą)
- Masyvo dydis nurodomas jį sukuriant ir nebegali kisti. Jį galima sužinoti per savybę „length“

One Dimensional array

Initialization

```
[int a[] = new int [12];]
```

Value

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

Index

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
------	------	------	------	------	------	------	------	------	------	-------	-------

```
System.out.print(a[5]);
```

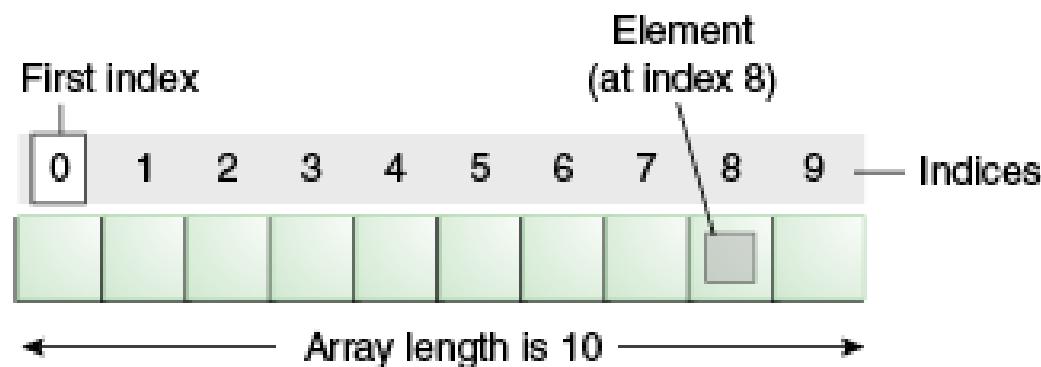
Output: 6

Deklaravimas

```
int[] marks;  
String[] words;  
double[] numbers;  
  
int primes[]; //legalu, bet nerekomenduojama
```

Inicjalizavimas

```
//dataType[] arrayRefVar = new dataType[arraySize];  
  
int[] arr = new int[10]; //Sukurs tuščią masyvą 10čiai skaičių saugoti  
  
//dataType[] arrayRefVar = {value0, value1, ..., valuek};  
  
int[] primes = {1,2,3,5,6,11,13}; //Sukurs 7 elementų masyvą ir užpildys ji  
//nurodytomis reikšmėmis būtent tokią tvarka
```



Reikšmės pagal nutylėjimą

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Reikšmių priskyrimas

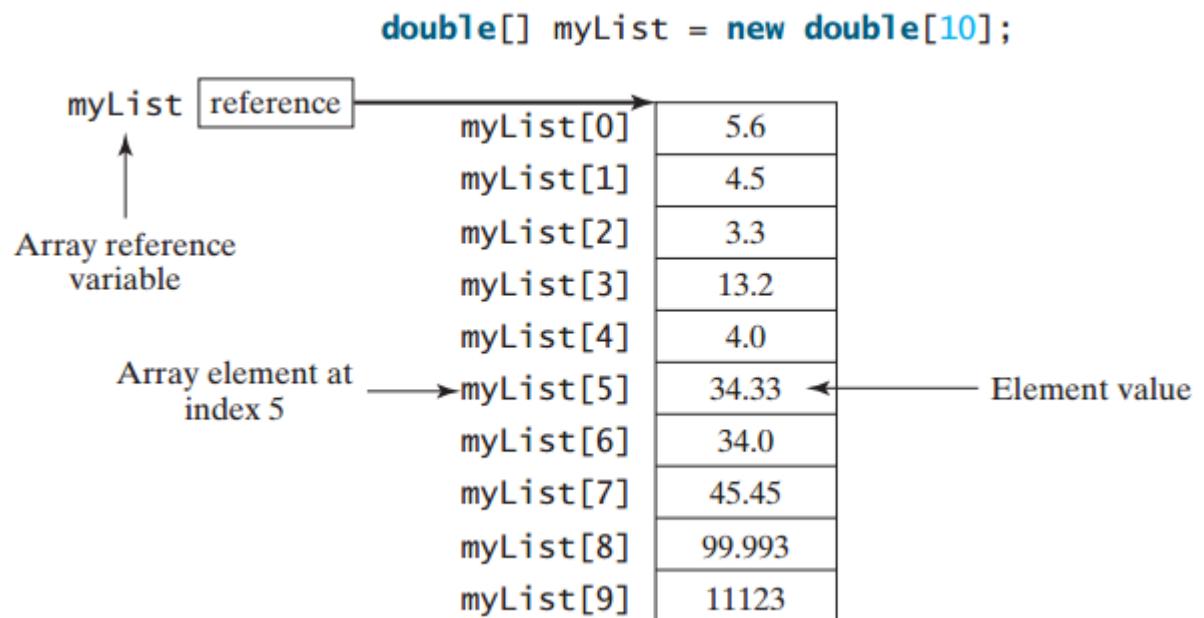
```
// Declare & allocate a 5-element array
int[] marks = new int[5];

// Assign values to the elements
marks[0] = 95;
marks[1] = 85;
marks[2] = 77;
marks[3] = 69;
marks[4] = 66;

System.out.println(marks[0]); // 95
System.out.println(marks[3] + marks[4]); // 135
```

Reikšmių priskyrimas

```
myList[0] = 5.6;  
myList[1] = 4.5;  
myList[2] = 3.3;  
myList[3] = 13.2;  
myList[4] = 4.0;  
myList[5] = 34.33;  
myList[6] = 34.0;  
myList[7] = 45.45;  
myList[8] = 99.993;  
myList[9] = 11123;
```



Masyvo elementų spausdinimas

```
double[] myList = { 1.9, 2.9, 3.4, 3.5 };

// Print all the array elements
for (int i = 0; i < myList.length; i++) {
    System.out.println(myList[i] + " ");
}
```

Masyvo elementų suma

```
double[] myList = { 1.9, 2.9, 3.4, 3.5 };

// Summing all elements
double total = 0;
for (int i = 0; i < myList.length; i++) {
    total = total + myList[i];
}
System.out.println("Total is " + total); // 11.7
```

Didžiausios reikšmės paieška

```
double[] myList = { 1.9, 2.9, 3.4, 3.5 };

// Finding the largest element
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max)
        max = myList[i];
}
System.out.println("Max is " + max); // 3.5
```

Elementų, tenkinančių sąlygą, kiekis

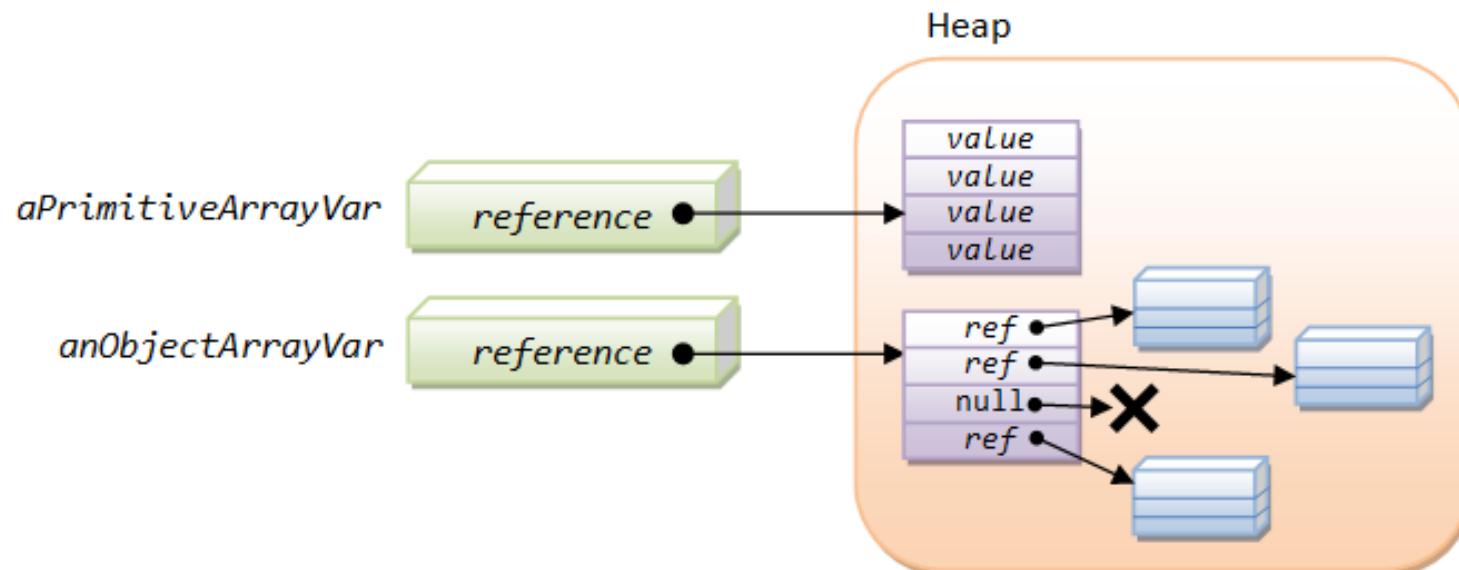
```
double[] myList = { 1.9, 2.9, 3.4, 3.5 };

int count = 0;
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > 3)
        count++;
}
System.out.println("Count: " + count); // 2
```

enhanced for-loop

```
for (int i=0; i < array.length; i++) {  
    system.out.println("Element: " + array[i]);  
}  
  
//enhanced for-loop  
  
for (String element : array) {  
    system.out.println("Element: " + element);  
}
```

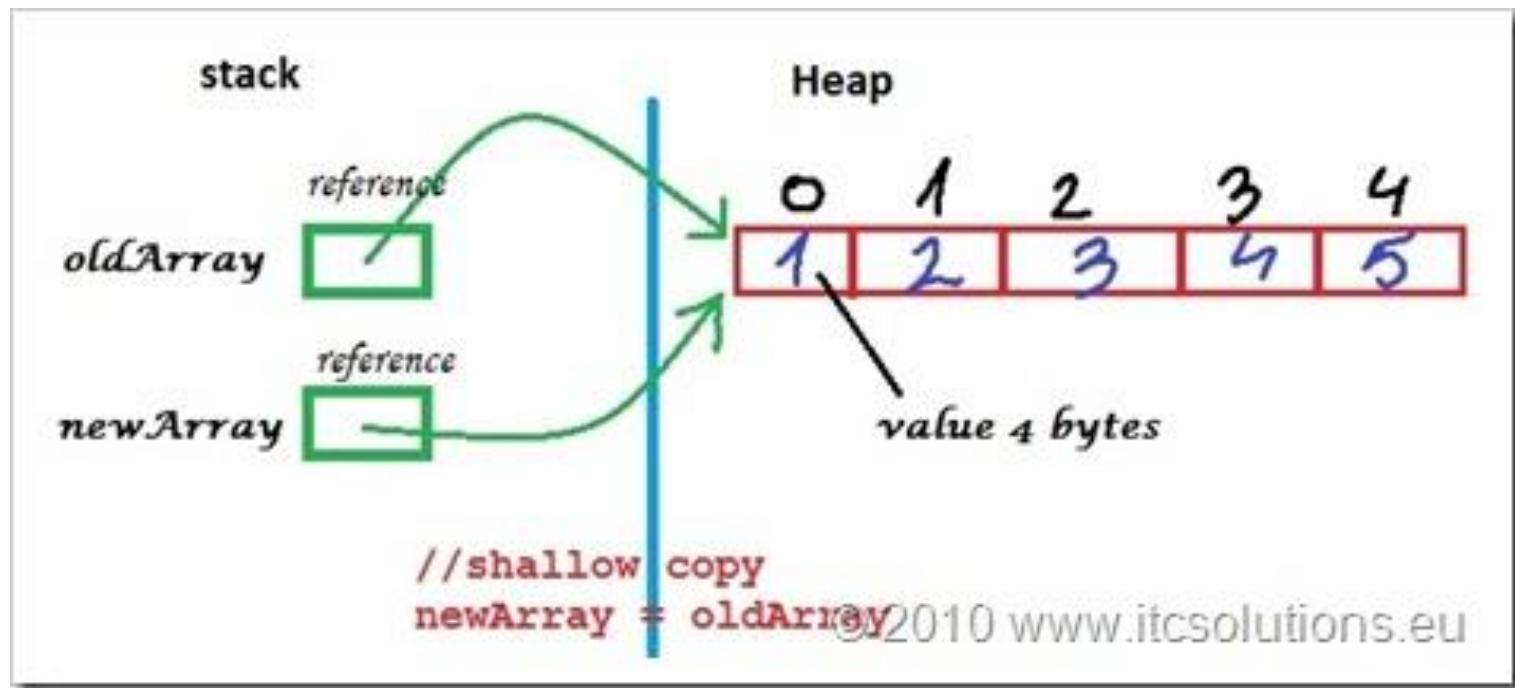
Masyvai



Arrays.toString

```
int[] arr = { 1, 2, 3, 4, 5 };  
  
System.out.println(arr); //prints [I@15db9742  
  
System.out.println(Arrays.toString(arr));  
//prints [1, 2, 3, 4, 5]
```

Kopijavimas



System.arraycopy

```
int[] arr = { 1, 2, 3, 4, 5 };

int[] copied = new int[10];
// arraycopy(sourceArray, srcPos,targetArray, tarPos, length)
System.arraycopy(arr, 0, copied, 1, 5);

System.out.println(Arrays.toString(arr));
System.out.println(Arrays.toString(copied));
```

```
[1, 2, 3, 4, 5]
[0, 1, 2, 3, 4, 5, 0, 0, 0, 0]
```

Daugiamatis masyvas

```
//declare
int[][] multiArr;

//instantiate
int[][] arr=new int[3][3];//3 row and 3 column

//initialize
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
//...
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;

//declaring and initializing 2D array
int arr2[][]={{1,2,3},{2,4,5},{4,4,5}};
```

Daugiamatis masyvas

	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix = new int[5][5];
```

	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	7	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix[2][1] = 7;
```

	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

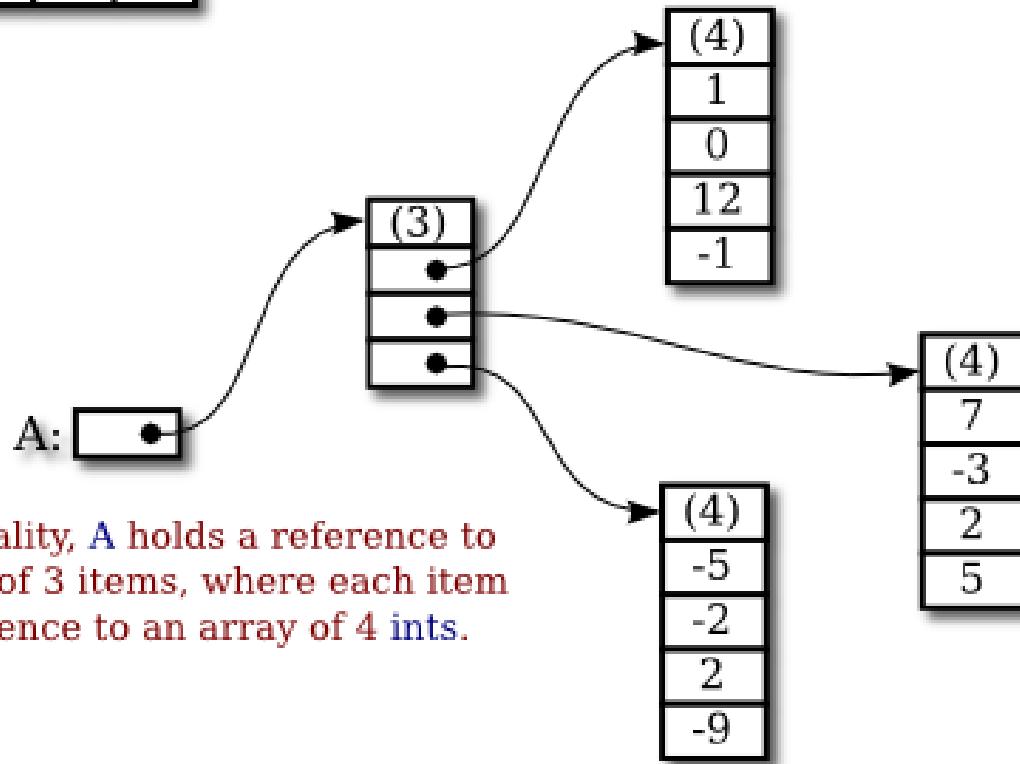
```
int[][] array = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};
```

Daugiamatis masyvas

A:

1	0	12	-1
7	-3	2	5
-5	-2	2	-9

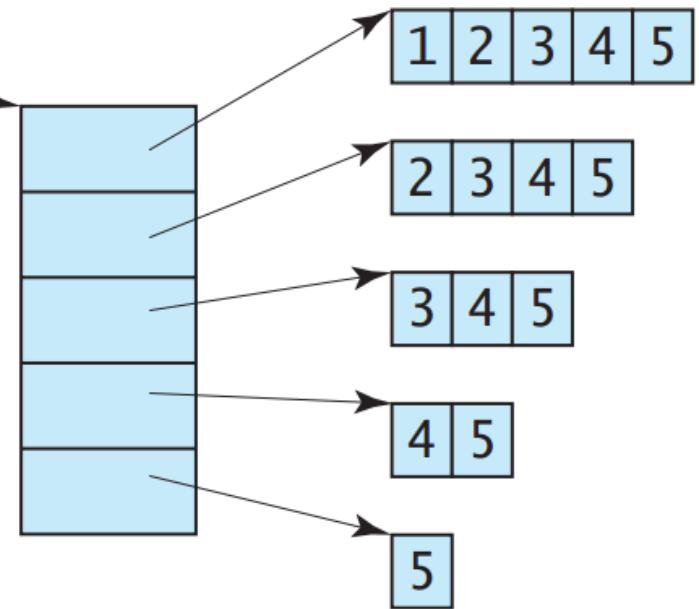
If you create an array `A = new int[3][4]`,
you should think of it as a "matrix" with
3 rows and 4 columns.



But in reality, `A` holds a reference to
an array of 3 items, where each item
is a reference to an array of 4 ints.

Ragged array

```
int[][] triangleArray = {  
    {1, 2, 3, 4, 5},  
    {2, 3, 4, 5},  
    {3, 4, 5},  
    {4, 5},  
    {5}  
};
```



Dvimačio masyvo spausdinimas

```
int arr[][] = { { 1, 2, 3 }, { 2, 4, 5 }, { 4, 4, 5 } };  
  
//printing 2D array  
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 3; j++) {  
        System.out.print(arr[i][j] + " ");  
    }  
    System.out.println();  
}
```

1	2	3
2	4	5
4	4	5

Dvimačio masyvo spausdinimas (2)

```
int arr[][] = { { 1, 2, 3 }, { 2, 4, 5 }, { 4, 4, 5 } };  
  
System.out.println(arr);  
//prints [I@15db9742  
  
System.out.println(Arrays.toString(arr));  
//prints [[I@6d06d69c, [I@7852e922, [I@4e25154f]  
  
System.out.println(Arrays.deepToString(arr));  
//prints [[1, 2, 3], [2, 4, 5], [4, 4, 5]]
```



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

7- ALGORITMAI

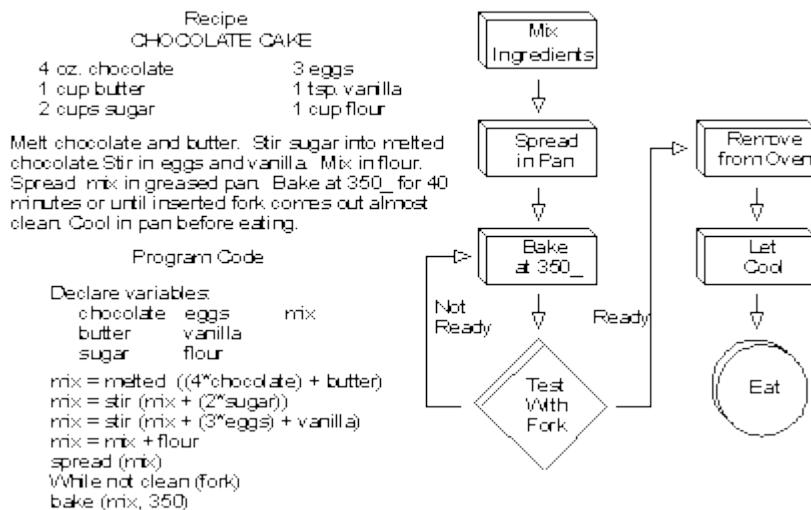
Jaroslav Grablevski / Justina Balsė

Algoritmas

- Algoritmas – tai veiksmų seka ar metodas, kuris naudojamas siekiant išspręsti užduotį ar uždavinį.
- Algoritmas gali būti suprantamas kaip funkciją, kuri apdoroja įėjimo duomenis ir gražina rezultatus.

Algoritmas

- Algoritmai užrašomas:
 - Schemomis (srautų diagrama)
 - Abstrakčia kalba (pseudokodu)
 - Normalia kalba
- Algoritmas realizuojamas viena iš programavimo kalbų



Pirminio skaičiaus nustatymas

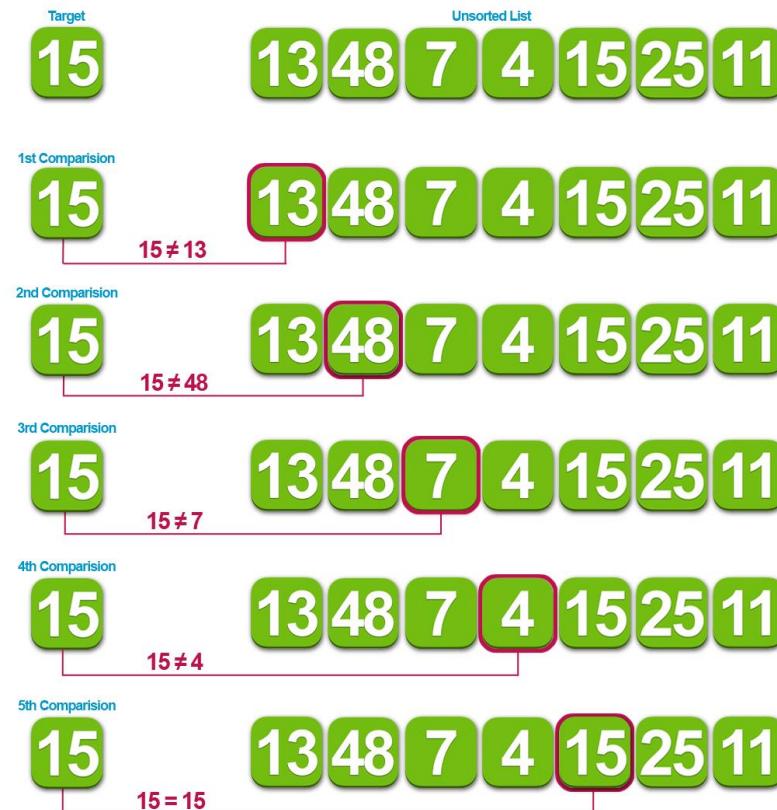
```
public static boolean arPirminis(long skaicius) {  
    for (long i = 2; i < skaicius - 1; i++) {  
        if (skaicius % i == 0)  
            return false;  
    }  
    return true;  
}
```

Pirminio skaičiaus nustatymas

```
public static boolean arPirminis2(long skaicius) {  
    if (skaicius % 2 == 0)  
        return false;  
    long max = (long) Math.floor(Math.sqrt(skaicius));  
    for (int j = 3; j <= max; j += 2) {  
        if (skaicius % j == 0)  
            return false;  
    }  
    return true;  
}
```

Nuosekli paieška (linear or sequential search)

- Primityviausias reikiama raktu paieškos masyve būdas yra pradėti nuo masyvo pradžios ir peržiūrėti iš eilės kiekvieną elementą



Dvejetainė paieška (binary search)

- **Dvejetainė paieška** yra paieškos sutvarkytame sąraše procesas. Jis dalina sutvarkytą sąrašą tol, kol atsiduria vietoje, kurioje yra ieškomas elementas, jei jis iš viso sąraše yra.

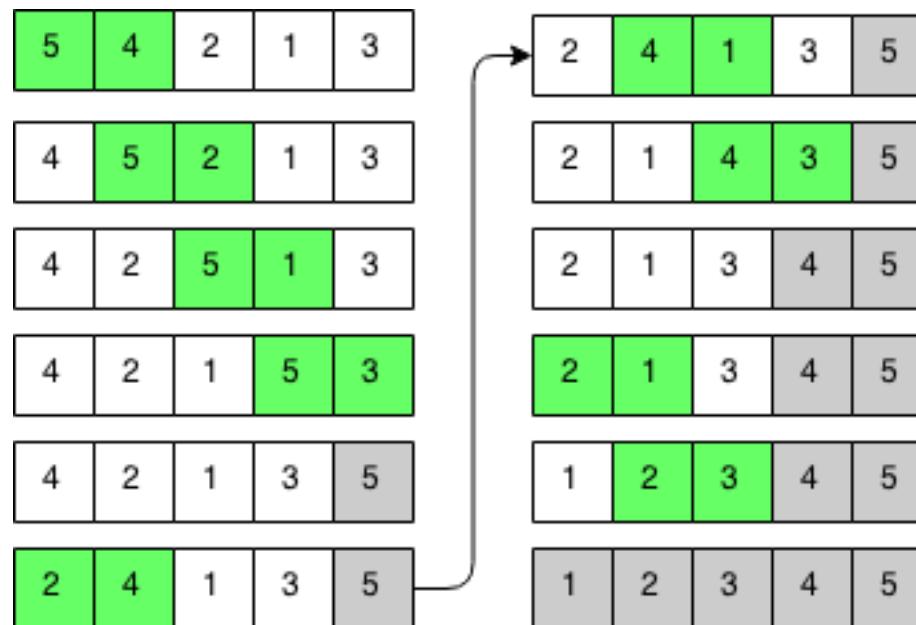


Rikiavimo algoritmai.

- **Rikiavimo algoritmas** – algoritmas, dėstantis duomenis tam tikra tvarka.
- Rikiavimo algoritmai gali būti skirstomi keliais būdais:
 - Pagal sudėtingumą.
 - Pagal naudojamą atmintį.
 - Pagal stabilumą.
- Daugiau informacijos:
 - <http://bfy.tw/20gG>
 - <https://www.toptal.com/developers/sorting-algorithms>
 - <https://visualgo.net/bn/sorting?slide=1>
 - [Java Algorithms \(Derek Banas\)](#)

Bubble sort

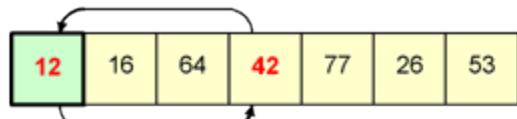
- Pagrindinė algoritmo idėja yra lyginti gretimus elementus ir, jeigu reikia, keisti juos vietomis. Kiekvienos iteracijos metu ne tik didžiausias iš nesurūšiuotų elementų padedamas į savo vietą, bet ir atliekamas kitų elementų patvarkymas.



Rūšiavimas paieška (*selection sort*)



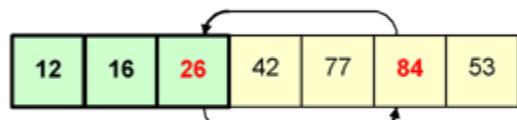
The array, before the selection sort operation begins.



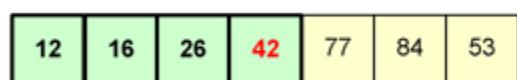
The smallest number (**12**) is swapped into the first element in the structure.



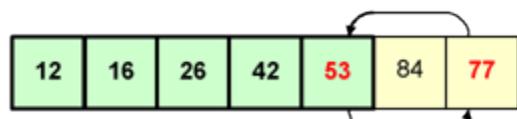
In the data that remains, **16** is the smallest; and it does not need to be moved.



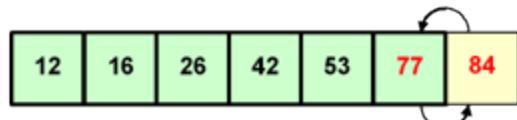
26 is the next smallest number, and it is swapped into the third position.



42 is the next smallest number; it is already in the correct position.

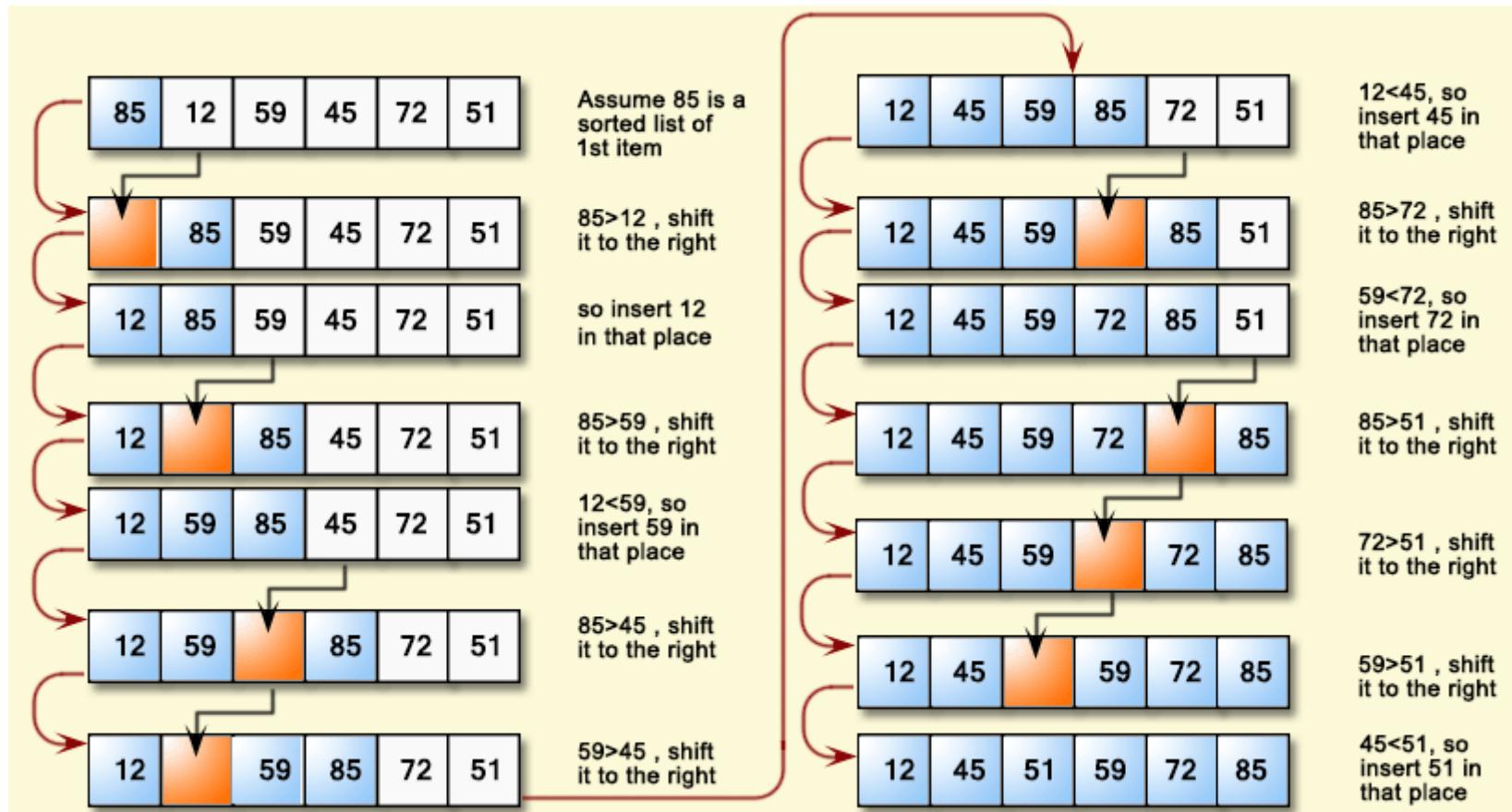


53 is the smallest number in the data that remains; and it is swapped to the appropriate position.



Of the two remaining data items, **77** is the smaller; the items are swapped. *The selection sort is now complete.*

Rūšiavimas įterpimu (*insertion sort*)



Greitasis rikiavimas (*quicksort*)

- Pasirinkti vieną masyvo elementą (bet kurį), šis elementas vadinamas „pivot“ elementu;
- 2. Likusius elementus suskirstyti į dvi grupes: a) kurie yra mažesni už pivot elementą, b) kurie yra didesni už pivot elementą;
- 3. Rekursyviai rūšiuoti kiekvieną grupę atskirai.

STEP 1: choose a pivot



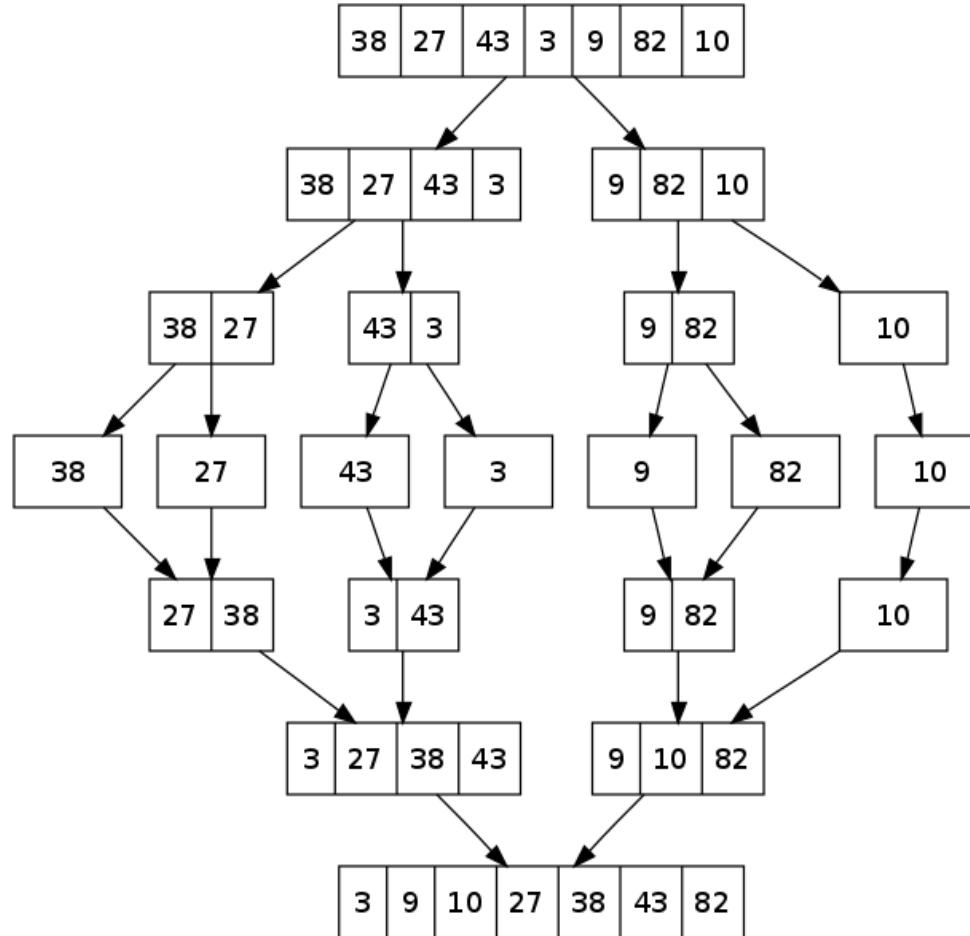
STEP 2: lesser values go to the left, greater values go to the right



STEP 3: repeat from step 1 with the two sub-lists

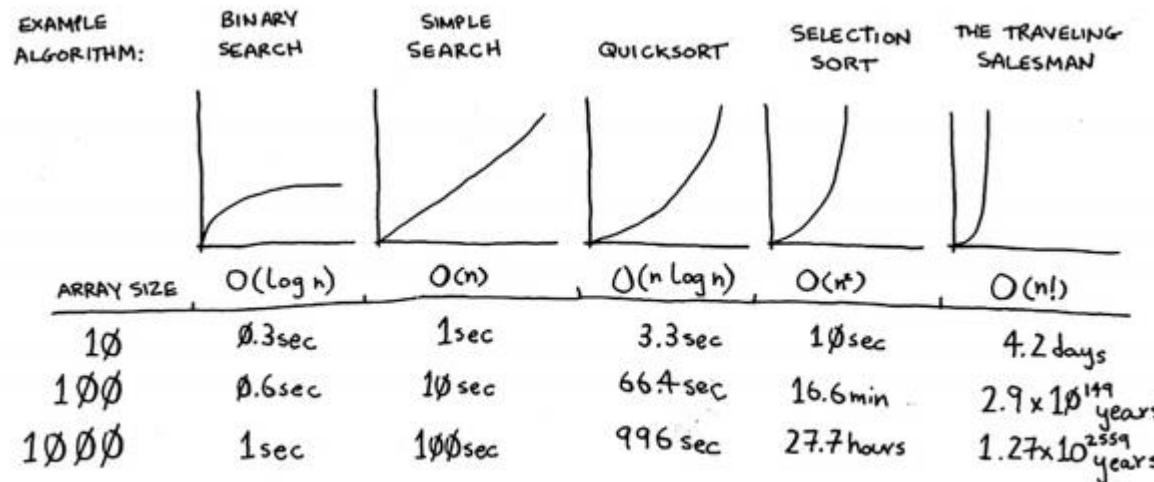


Rikiavimas sujungimu (*merge sort*)



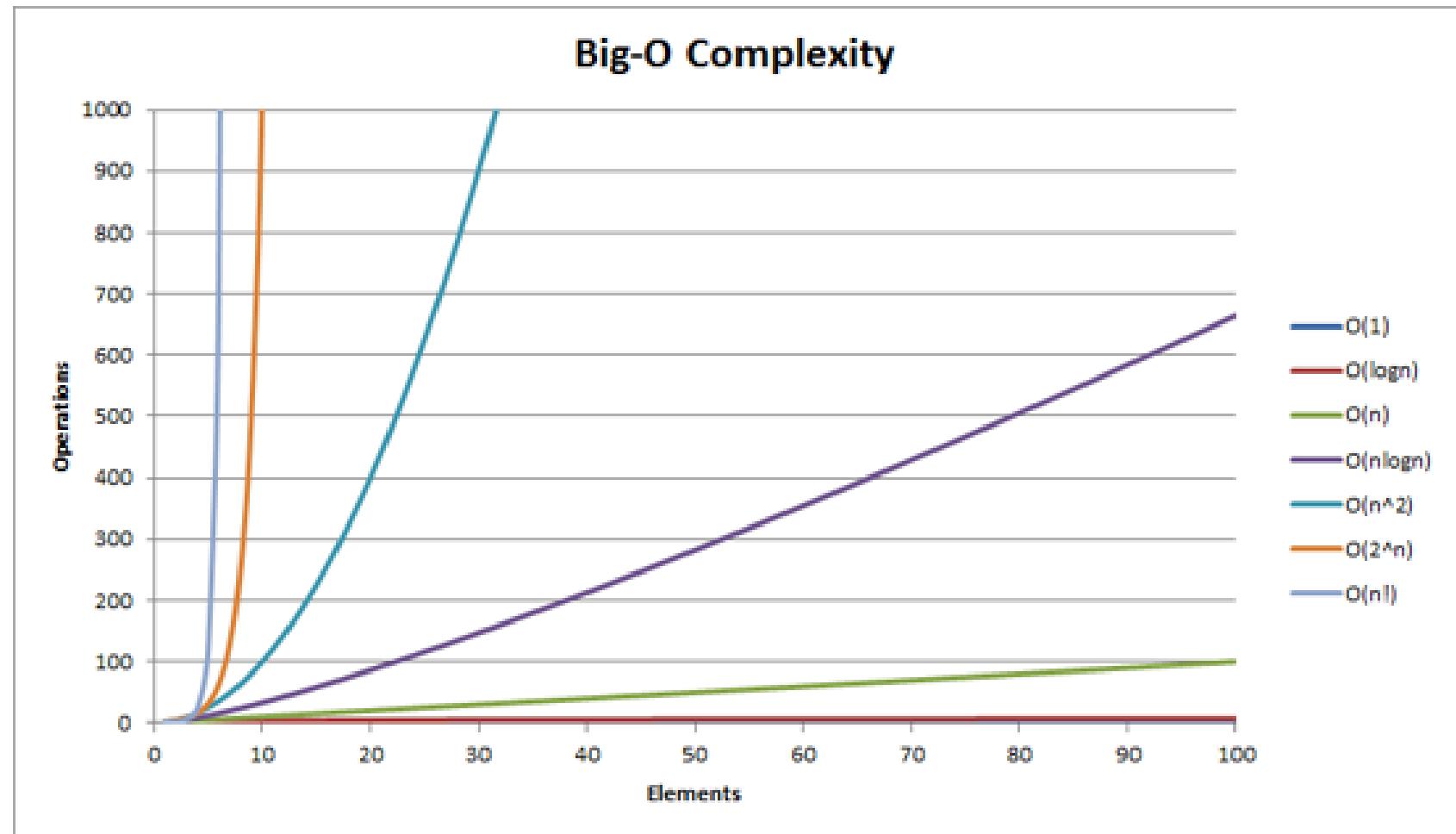
Algoritmu laiko sudėtingumas

- Laiko sudėtingumo skaičiavimas vertina, kiek laiko reiktu tam tikrai problemai su tam tikru duomenų dydžiu spręsti efektyviausiu algoritmu.
- Tarkime, kad turint n bitų duomenų kiekj, problema išsprendžiama per n^2 žingsnių; tokia problema yra n^2 sudėtingumo.



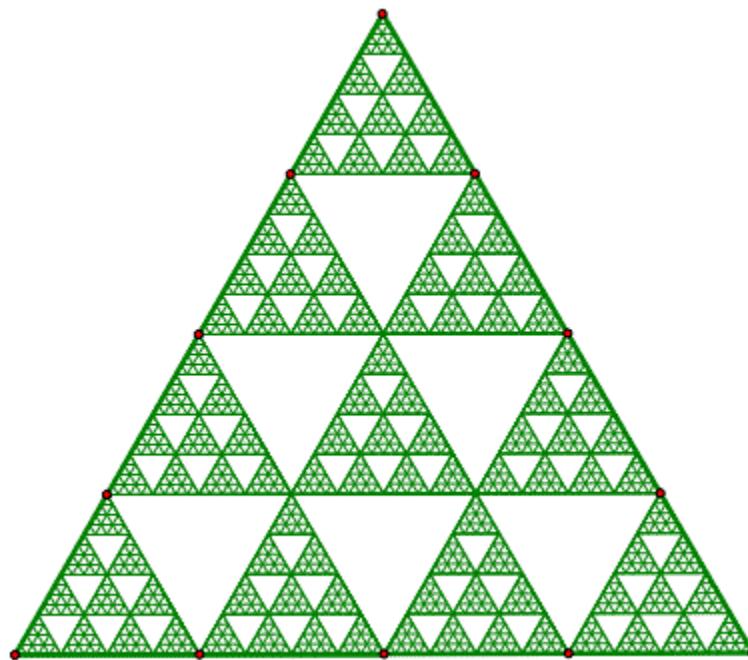
Estimates based on a slow computer that performs 10 operations per second

Big O notation



Rekursija

- Programų ar algoritmų sudarymo metodas, kai programa kreipiasi pati į save, esant mažesnėms argumentų reikšmėms.

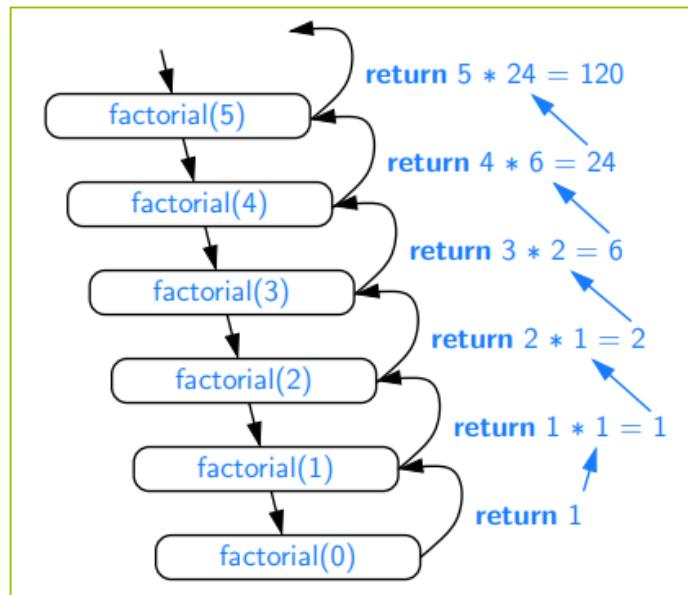


Rekursija

- Algoritmas vadinamas rekursiniu, jei jis kviečia save patj atlikti dalj skaičiavimų
- Rekursinis algoritmas privalo turēti dvi dalis:
 - Bazinę dalj, kuri sprendžiama nenaudojant rekursijos;
 - Rekursinę dalj, kurioje atliekamas tos pačios funkcijos kvietimas.
- Kiekvienoje rekursinėje procedūroje turi būti numatyti visi ribiniai atvejai, kuriuos pasiekus rekursija nutraukiamā.
- **Rekursijos gylis** – tai ilgiausia procedūrų grandinė, kuri suskaičiuoja funkcijos reikšmę.

Rekursija (faktorialas)

```
public static long factorial(long number) {  
    if (number <= 1) // test for base case  
        return 1; // base cases: 0! = 1 and 1! = 1  
    else // recursion step  
        return number * factorial(number - 1);  
}
```



Rekursija

- Rekursijos privalumai
 - Patogiai kontroliuojama procedūros eiga
 - Paprastas kodas
- Rekursijos trūkumai
 - Stack overflow
 - Nevisada efektyvus algoritmas
- Rekomendacija
 - Nenaudoti rekursijos, jei nežinomas elementų skaičius.



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

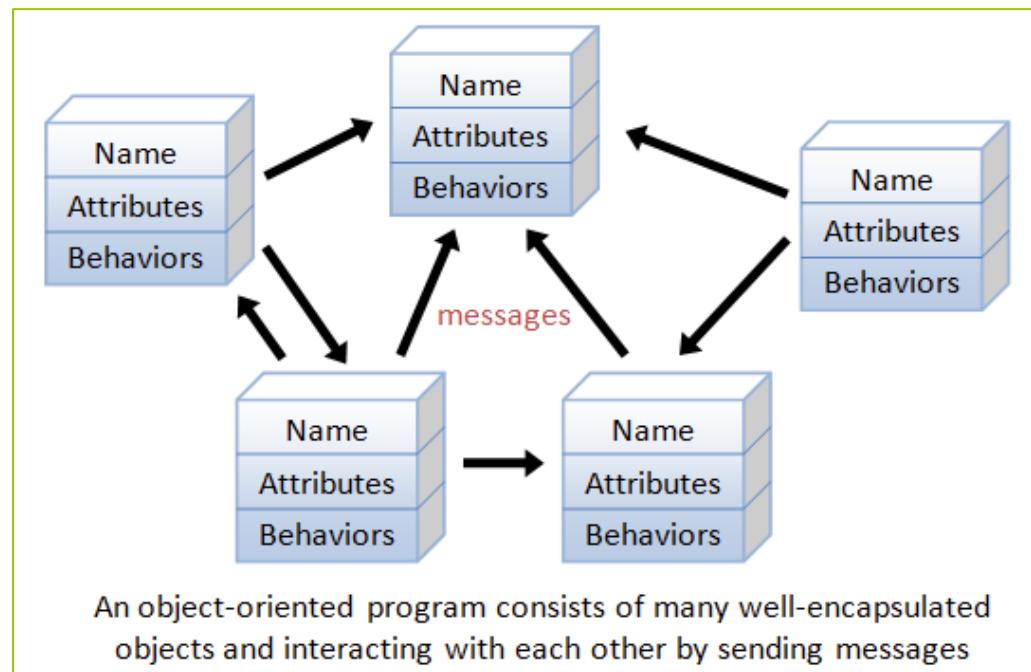
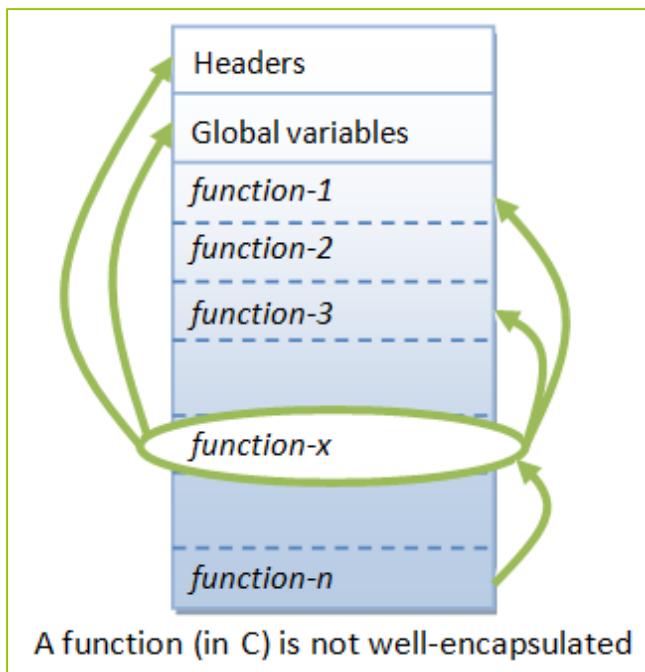
8 - OBJEKTINIS PROGRAMAVIMAS

Jaroslav Grablevski / Justina Balsė

Turinys

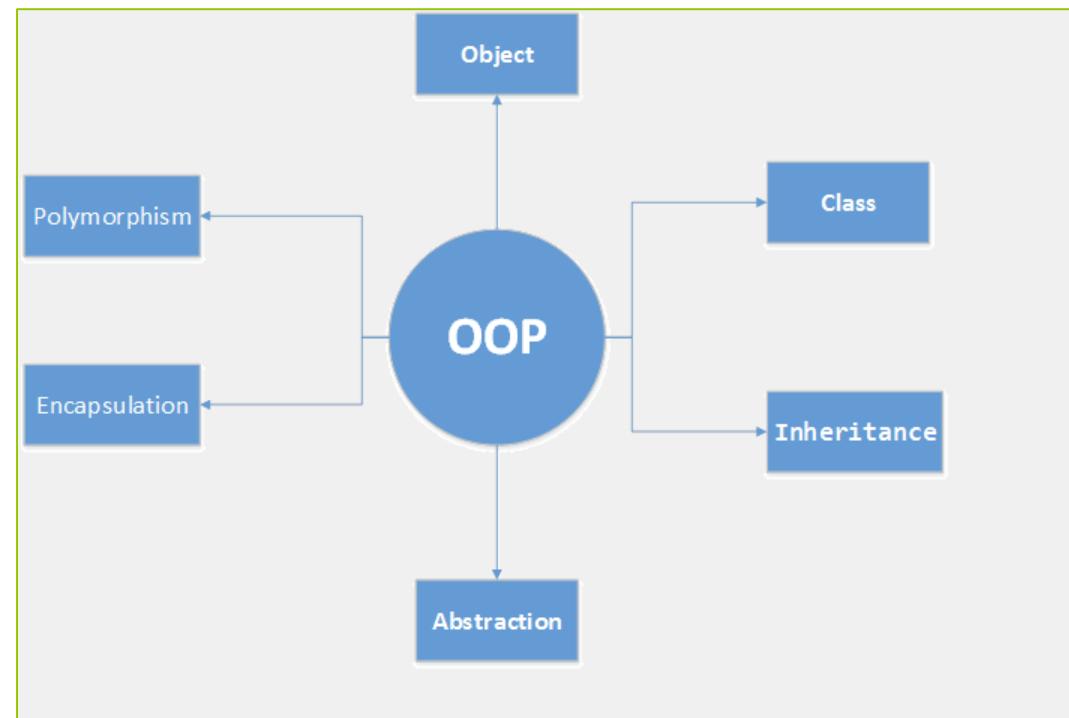
- Pagrindinės sąvokos
- Objektas
- Klasė
- Kintamieji
- Metodai
- Konstruktoriai
- Inkapsuliacija

Procedural vs. Object-Oriented



Pagrindinės OOP sąvokos ir terminai

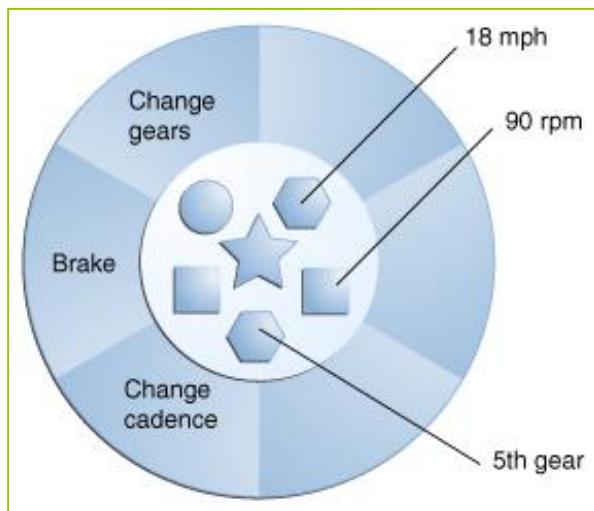
- Klasė
- Objektas
- Abstrakcija
- Inkapsuliacija
- Paveldėjimas
- Polimorfizmas



Objektas

Tai savarankiškas programinis komponentas turintis šias savybes:

- Tapatybė (*identity (or name)*)
- Būsena (**state** (*or attributes, variables, fields*))
- Elgesys (**behavior** (*or methods*))



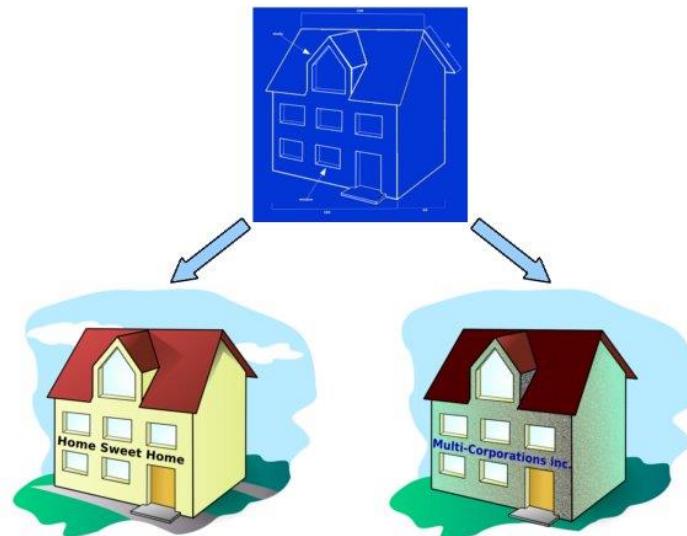
Bicycles have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes)

Abstrakcija (abstraction)

- Tai yra apibendrinimo principas, kuris nusako, kad konkrečius dalykus galime apibendrinti, išskiriant esmines ir pačias svarbiausias jų savybes bei atsiribojant nuo specifinių detalių.
- Tai yra viena iš pagrindinių priemonių kovojant su programinės įrangos sudėtingumu.

Klasė

- Klasė – tai objekto šablonas (brėžinys), kurį sudaro duomenys (kintamieji, laukai) ir metodai.
- Iš vienos klasės galima sukurti kiek norima objektų, tačiau kiekvienas objektas yra sukurtas iš vienos konkrečios klasės.



Klasė

- Klasės kūnas susideda iš tokių elementų:
 - Objekto kintamųjų
 - Konstantų
 - Konstruktorių
 - Metodų
 - Statinių inicializavimo blokų
 - Inicializavimo blokų
 - Klasių
 - Interfeisų

*Nė vienas nėra privalomas

Klase

```
public class Bicycle {  
  
    // fields  
    int cadence;  
    int gear;  
    int speed;  
  
    // constructor  
    Bicycle(int cadence, int gear, int speed) {  
        this.cadence = cadence;  
        this.gear = gear;  
        this.speed = speed;  
    }  
  
    // method  
    public void starting(){  
    }  
}
```

Klasy

```
public class Charge
{
    private final double rx, ry;
    private final double q;

    public Charge(double x0, double y0, double q0)
    { rx = x0; ry = y0; q = q0; }

    public double potentialAt(double x, double y)
    {
        double k = 8.99e09;
        double dx = x - rx;
        double dy = y - ry;
        return k * q / Math.sqrt(dx*dx + dy*dy);
    }

    public String toString()
    { return q + " at (" + rx + ", " + ry + ")"; }

    public static void main(String[] args)
    {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);
        Charge c1 = new Charge(0.51, 0.63, 21.3);
        Charge c2 = new Charge(0.13, 0.94, 81.9);
        double v1 = c1.potentialAt(x, y);
        double v2 = c2.potentialAt(x, y);
        StdOut.printf("%.2e\n", (v1 + v2));
    }
}
```

Annotations pointing to code elements:

- instance variables*: points to `rx`, `ry`, and `q`.
- constructor*: points to the `Charge` constructor.
- instance methods*: points to the `potentialAt` and `toString` methods.
- test client*: points to the `main` method.
- create and initialize object*: points to the two `new Charge` constructor invocations.
- object name*: points to the variable `c1` and `c2`.
- class name*: points to the `Charge` class name.
- instance variable names*: points to `rx` and `ry` within the `potentialAt` method.
- invoke constructor*: points to the first `new Charge` invocation.
- invoke method*: points to the two `c1.potentialAt` and `c2.potentialAt` method invocations.

Objekto kintamieji (instance variables)

- Deklaruojami klasės ribose (bet ne metoduose, konstruktoriuose ar blokuose)
- Vadinami objekto laukais (*fields*)
- Priklauso konkrečiam klasės egzemplioriui/objektui (iekvienas objektas turi nuosavas šių kintamųjų kopijas)
- Atmintis šiems kintamiesiems išskiriama kuriant objektą
- Prieinami:
 - Metoduose;
 - Konstruktoriuose;
 - Blokuose;
- Galima naudoti prieinamumo modifikatorius
- Priskiriamos numatytosios reikšmės
- Pasiekiami naudojant: <objektoNuoroda>.kintamasis
- Objekto kintamieji nėra pasiekiami statiniams metodams

Klasės kintamieji (static class members)

- Deklaruojami klasės ribose (bet ne metoduose, konstruktoriuose ar blokuose)
- Priklauso konkrečiai klasei (viena kopija kintamųjų vienai klasei, nepriklausomai nuo to, kiek sukurta klasės objektų)
- Atmintis šiems kintamiesiems išskiriama programos startavimo metu
- Pasiekiami naudojant: <KlasėsPavadinimas>.kintamasis
- Dažniausiai naudojami kaip konstantos, prieinamos ir kitiems objektams

Kintamųjų galiojimo zona (scope)

- Kintamieji, paskelbti klasėje, galioja visai klasei, t.y. galioja visiems klasės metodams ir klasės vidinėms klasėms.
- Metodo kintamieji galioja tik tam metodui.
- Metodo parametrai galioja tik tam metodui.

Klasės kintamieji (static class members)

```
// Static variable used to maintain a count of the number of
// Employee objects in memory.

public class Employee {
    private static int count = 0; // number of Employees created
    private String firstName;
    private String lastName;

    // initialize Employee, add 1 to static count and
    // output String indicating that constructor was called
    public Employee(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;

        ++count; // increment static count of employees
    }

    // static method to get static count value
    public static int getCount() {
        return count;
    }
}
```

Konstantos

- Java programavimo kalboje konstanta yra realizuojama naudojant žodelį ***final***, kuris nusako, kad reikšmė, kuri yra priskirta nekintama.

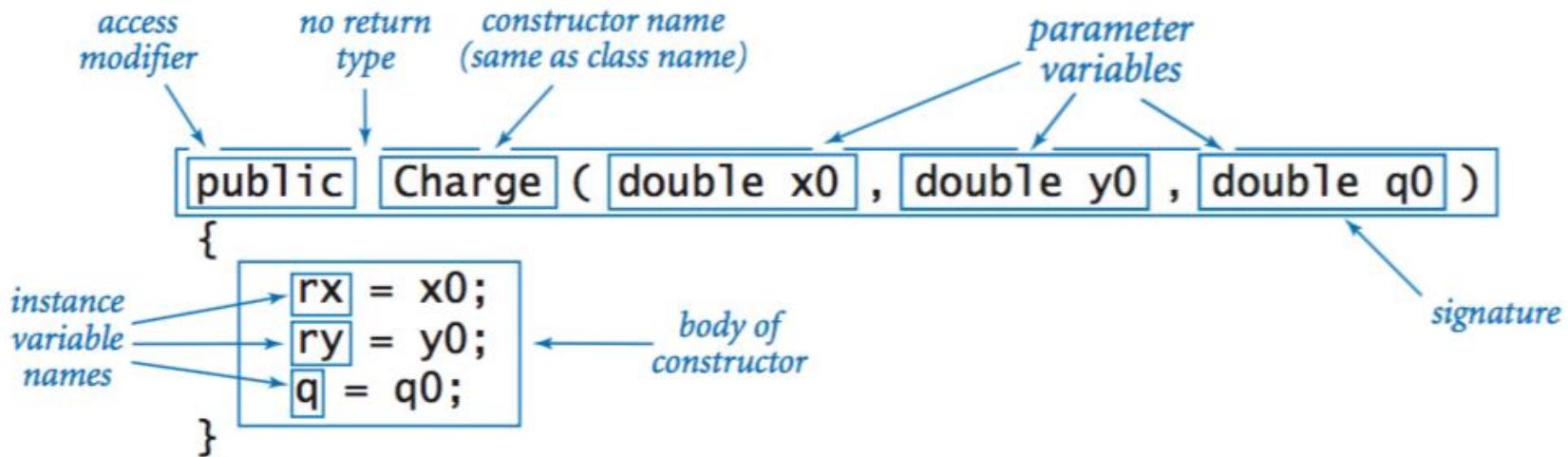
```
//klasės konstantos
public static final float PI = 3.14f;
public static final double G = 6.674 * Math.pow(10, -11);
public static final int C = 299_792_458;
public static final int ZERO = 0;

//objekto konstantos
private final String name;
private final int radius;
```

Konstruktorius

- Konstruktoriai skirti klasės inicializavimui - tai yra klasės būsenos paruošimui tuomet, kai sukuriamas objektas.
- Klasės konstruktoriai yra pagrindinė priemonė klasės objektams kurti.
- Panašus į metodą
- Vardas sutampa su klasės
- Neturi gražinamo tipo
- Gali būti keli (0..*)

Konstruktorius



Konstruktorius

```
// fields
int cadence;
int gear;
int speed;

// constructors
Bicycle() {
    this(0, 0, 0);
}

Bicycle(int cadence, int gear) {
    this.cadence = cadence;
    this.gear = gear;
}

Bicycle(int cadence, int gear, int speed) {
    this.cadence = cadence;
    this.gear = gear;
    this.speed = speed;
}
```

Konstruktorius

- Klasė gali turėti daug konstruktorių, kurie gali būti perkrauti (*overloaded*)
- Būtina sąlyga – konstruktoriai turi skirtis savo antraštėmis, t.y. parametru skaičiumi arba parametru tipais.

Konstruktorius > this

- Sakinys **this([<<argumentai>>])** yra naudojamas klasės konstruktoriuose iškvesti kitiems konstruktoriams.
- Šis sakinys turi būti pats pirmasis konstruktoriuje.

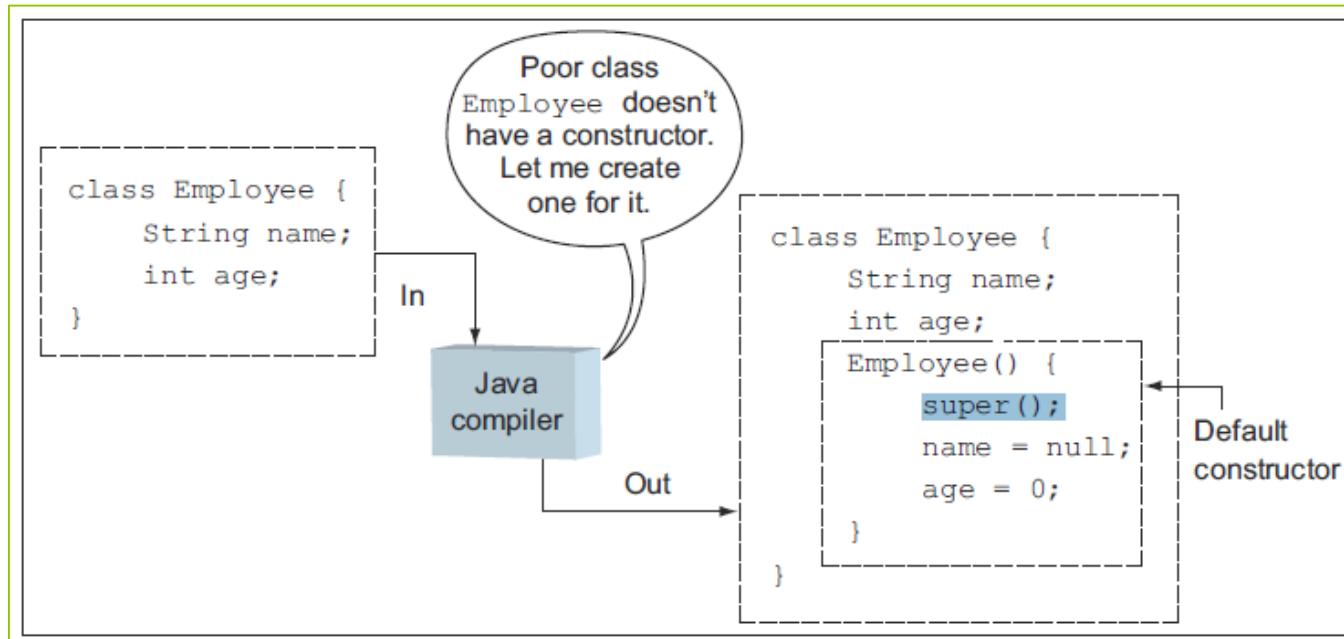
```
class Person{
    String name;
    int age;

    Person() {
        this("Anonymous", 0);
    }

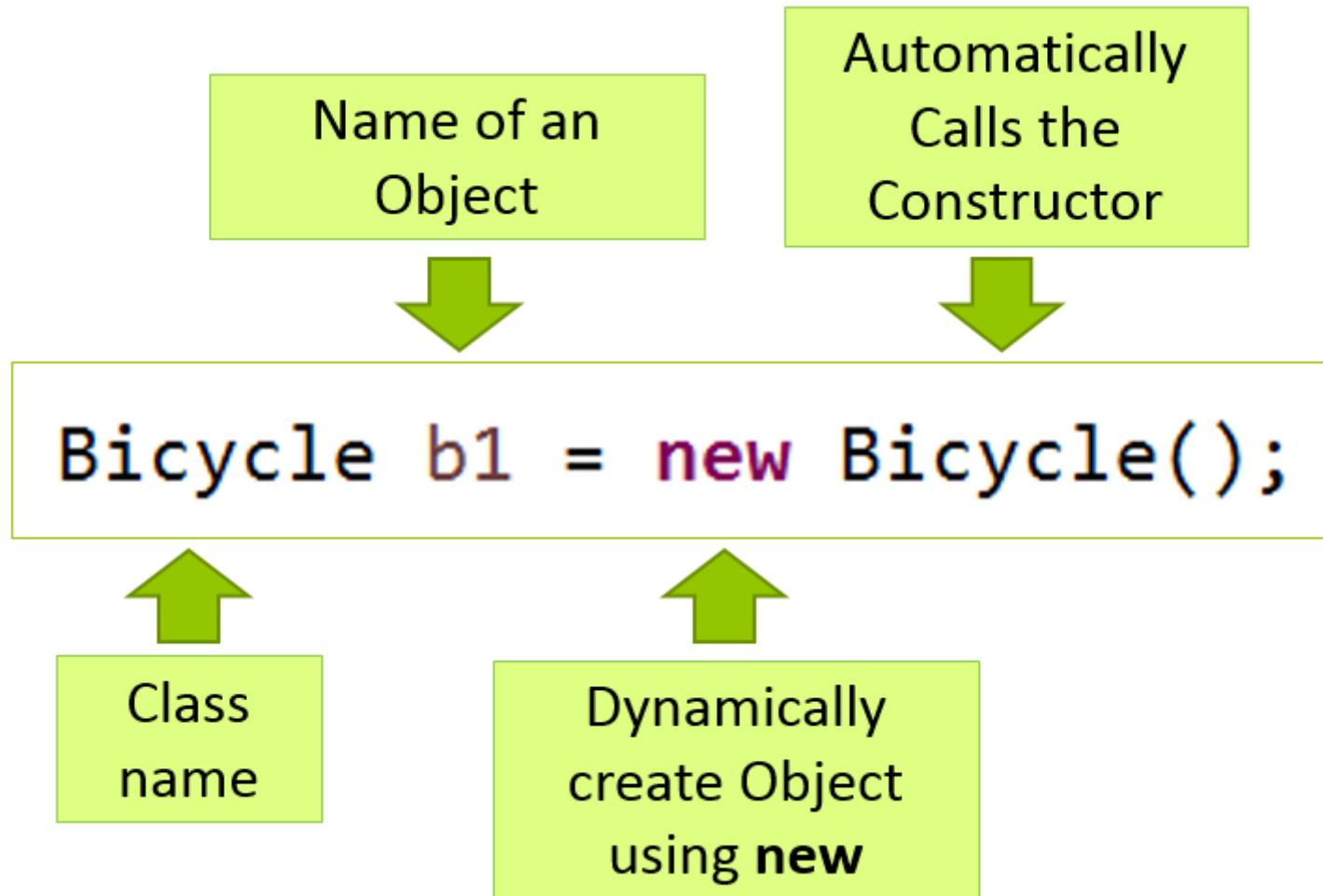
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Konstruktorius

- Jeigu konstruktorius nėra naudojamas, kuriant objektą kviečiamas numatytais konstruktorius, kuris visiems klasės kintamiesiems (laukams) suteikia „nulines“ reikšmes (pagal tipą).



Kaip sukurti objektą?



this

- Žodelis this, tai nuorodą į dabartinį objektą - nuoroda objekto, kurio metodas ar konstruktorius konkrečiu metu yra vykdomas.
- Dažniausiai naudojamas tais atvejais, kai metodo ar konstruktoriaus parametrų pavadinimai sutampa su objekto kintamujų vardais.
- Papildomai, jis gali būti naudojamas tose vietose, kur reikalinga dabartinio objekto nuoroda.

```
public class Point {  
    int x = 0;  
    int y = 0;  
  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Metodai

- Metodai gali būti dvieju tipų:
 - **Klasės** (statiniai) metodai
 - Priklauso klasei
 - Norint iškvesti, nebūtinės objekto sukūrimas - užtenka klasės pavadinimo: *<klasėsPavadinimas>.<metodoPavadinimas>([<<argumentai>>])*
 - Gali dirbti tik su klasės (statius) kintamaisiais
 - Statiniai metodai pažymimi žodeliu **static**
 - **Objekto** metodai
 - Priklauso objektui
 - Norint iškvesti, būtinės objekto sukūrimas
 - Gali dirbti tiek su klasės, tiek su objekto kintamaisiais

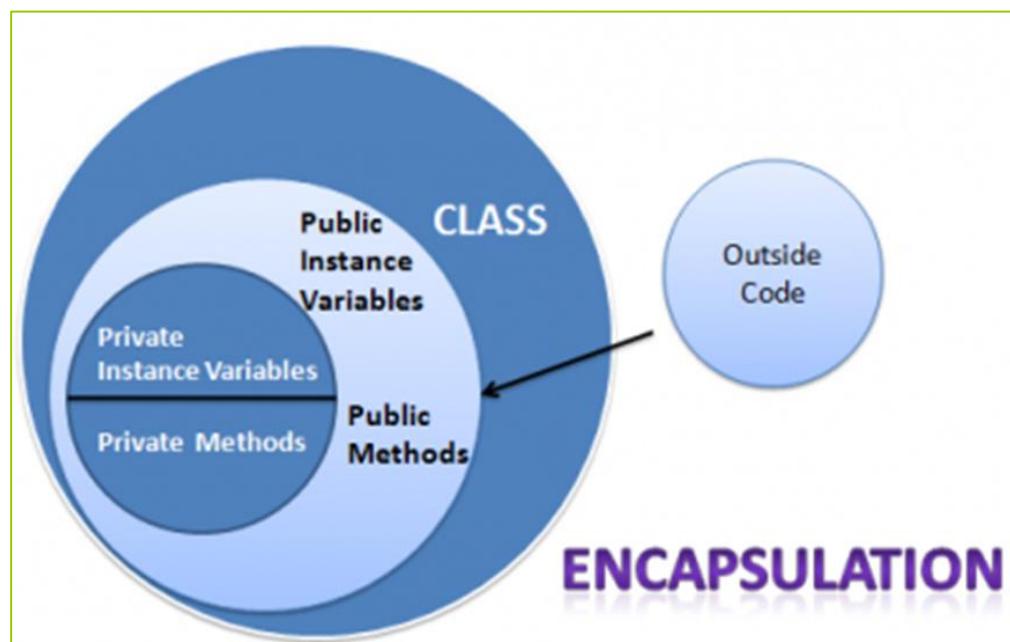
Metodai (overloading)

- Metodai gali būti perkrauti, t.y. klasėje gali būti deklaruoti daugiau nei vienas metodas su tuo pačiu pavadinimu.
- Svarbu, kad būtų tenkinamos šios taisyklės:
 - Parametru kiekis turi skirtis
 - Turi skirtis parametrų tipai

```
class Person{  
    String name;  
    int age;  
  
    void speak(){...}  
  
    void speak(int times){...}  
  
    void speak(String lastName){...}  
  
    void speak(int times, String lastName){...}  
}
```

Inkapsuliacija (encapsulation)

- Veiksmai ir duomenys supakuoti kartu
- Programuotojas renkasi ką rodyti, o ką slėpti nuo objekto naudotojo (kito programuotojo)



Paketas

- Paketas - tai vardų erdvė (angl. namespace), kurioje laikoma aibė susijusių klasių ir interfeisų. Paketai leidžia organizuoti programinius komponentus, panašiai, kaip katalogai leidžia organizuoti failus asmeniniam kompiuteryje.
- Papildomai, paketas leidžia slėpti tam tikras detales nuo kituose paketuose esančių klasių (inkapsuliacija), tokias kaip:
 - Klasės
 - Interfeisai
 - Kintamieji
 - Metodai

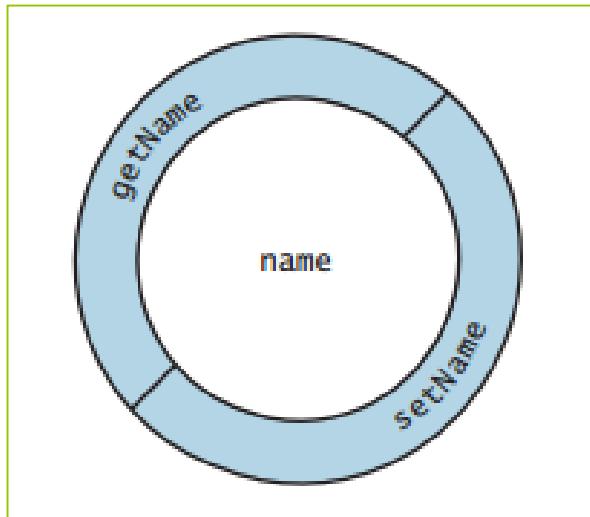
Modifikatoriai / access modifiers

- Kintamųjų ir metodų galiojimo zoną papildomai reguliuoja modifikatoriai.

Modifier	Same class or nested class	Other class inside the same package	Extended Class inside another package	Non-extended inside another package
private	yes	no	no	no
default (package private)	yes	yes	no	no
protected	yes	yes	yes	no
public	yes	yes	yes	yes

Set ir Get metodai

- Klasēs metodai, prasidedantys priešdēliais **get** ir **set** (**is** ir **set** boolean tipo kintamiesiems), skirti objekto savybēms nuskaityti ir pakeisti.



```
class Person{  
    private String name;  
    private int age;  
  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getAge() {  
        return age;  
    }  
    public void setAge(int age) {  
        this.age = age;  
    }  
}
```

Set ir Get metodai

```
public class Example3 {  
  
    public static void main(String[] args) {  
  
        Person p1 = new Person();  
  
        p1.setName("William");  
        p1.setAge(31);  
  
        System.out.println(p1.getName() + " is " + p1.getAge() + " years old.");  
    }  
}
```

William is 31 years old.

Metodos `toString()`

```
class Person{  
}  
  
public class ExampleToString {  
  
    public static void main(String[] args) {  
        Person p1 = new Person();  
        System.out.println(p1);  
    }  
}
```

Person@15db9742

Metodos `toString()`

```
class Person{  
  
    private String name;  
    private int age;  
  
    public Person(String name, int age){  
        this.name = name;  
        this.age = age;  
    }  
  
    @Override  
    public String toString() {  
        return name + " is " + age + " years old.";  
    }  
}  
  
public class ExampleToString {  
  
    public static void main(String[] args) {  
        Person p1 = new Person("Wiliam", 31);  
        System.out.println(p1);  
    }  
}
```

Wiliam is 31 years old.



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

9 - OBJEKTINIS PROGRAMAVIMAS 2

Jaroslav Grablevski / Justina Balsė

Paveldėjimas (inheritance)

- Tai klasės gebėjimas paveldėti protėvių klasės duomenis (kintamuosius, laukus) ir metodus.
- Klasė, kuri yra išvesta iš kitos klasės vadina poklase arba **vaikine** (angl. *subclass, derived class, extended class* arba *child class*).
- Klasė, iš kurios poklasė yra išvesta, vadina superklase arba **tėvine** (angl. *superclass, base class, parent class*).

Paveldėjimas

- Tai klasės gebėjimas paveldėti protėvių klasės duomenis (kintamuosius, laukus) ir metodus.
- Tėvinė klasė apibrėžia tam tikrus kintamuosius bei metodus, kuriuos vaikinė klasė paveldi.
- Vaikinė klasė gali pasipildyti savais kintamaisiais ir metodais
- Vaikinė klasė turi galimybę pakeisti tėvinių metodų veikimą juos perrašant (angl. *override*)

Paveldėjimas

- Java kalboje paveldėjimo ryšys deklaruojamas naudojant žodelį **extends** po kurio nurodoma klasė iš kurios yra paveldima

```
public class Car extends Machine {  
    // ...  
}
```

Paveldėjimas



Programmer

```
name  
address  
phoneNumber  
experience  
programmingLanguages  
writeCode()
```



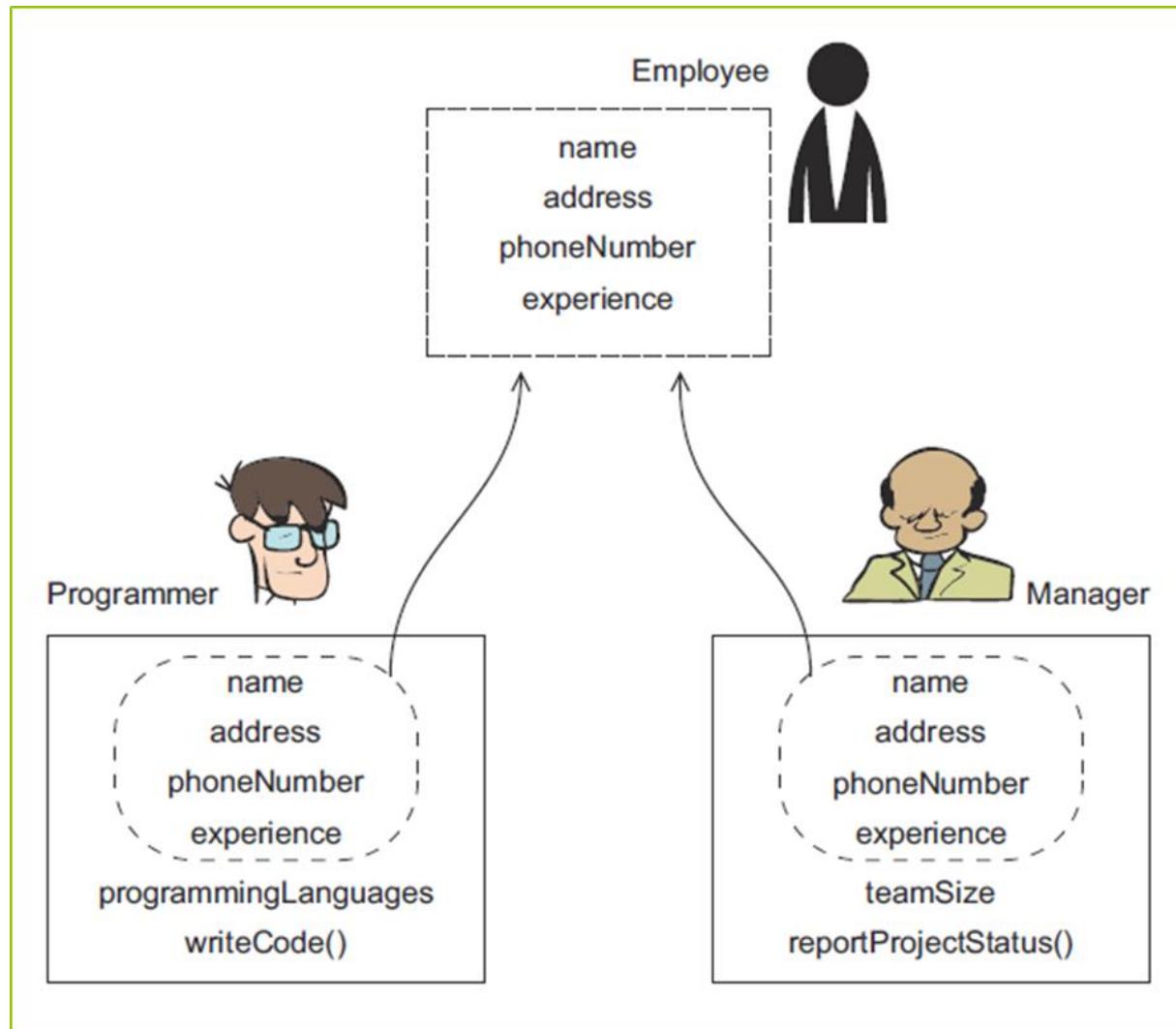
Manager

```
name  
address  
phoneNumber  
experience  
teamSize  
reportProjectStatus()
```

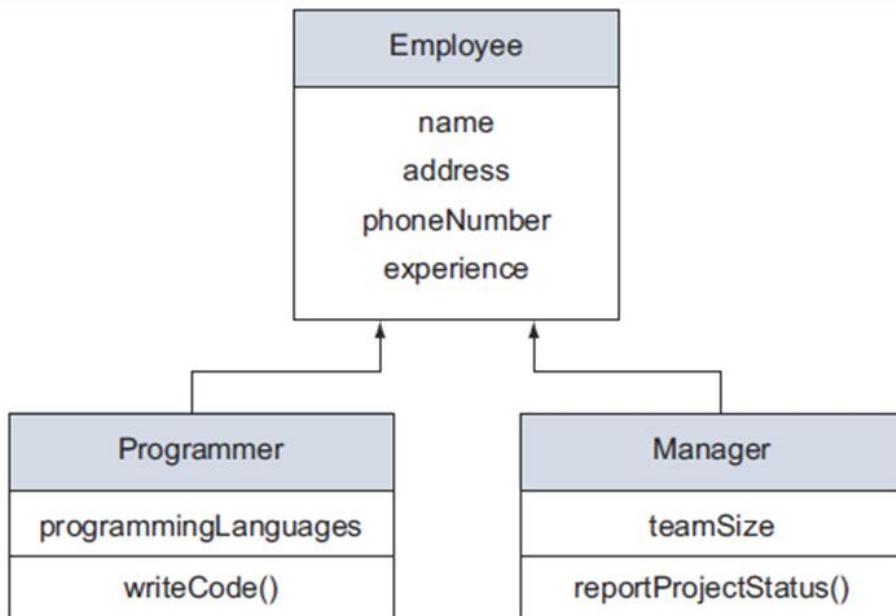
```
class Programmer {  
    String name;  
    String address;  
    String phoneNumber;  
    float experience;  
    String[] programmingLanguages;  
    void writeCode() {}  
}
```

```
class Manager {  
    String name;  
    String address;  
    String phoneNumber;  
    float experience;  
    int teamSize;  
    void reportProjectStatus() {}  
}
```

Paveldējimas



Paveldējimas

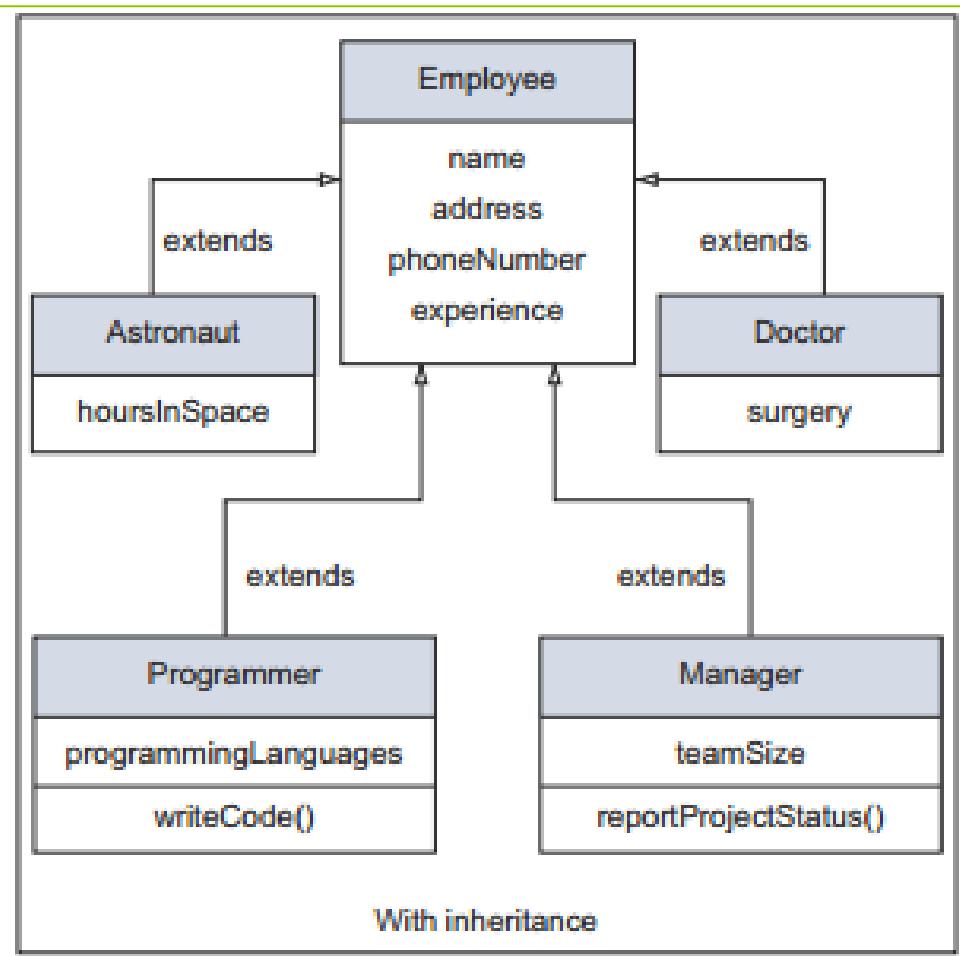


```
class Employee {
    String name;
    String address;
    String phoneNumber;
    float experience;
}
class Programmer extends Employee {
    String[] programmingLanguages;
    void writeCode() {}
}
class Manager extends Employee {
    int teamSize;
    void reportProjectStatus() {}
}
```

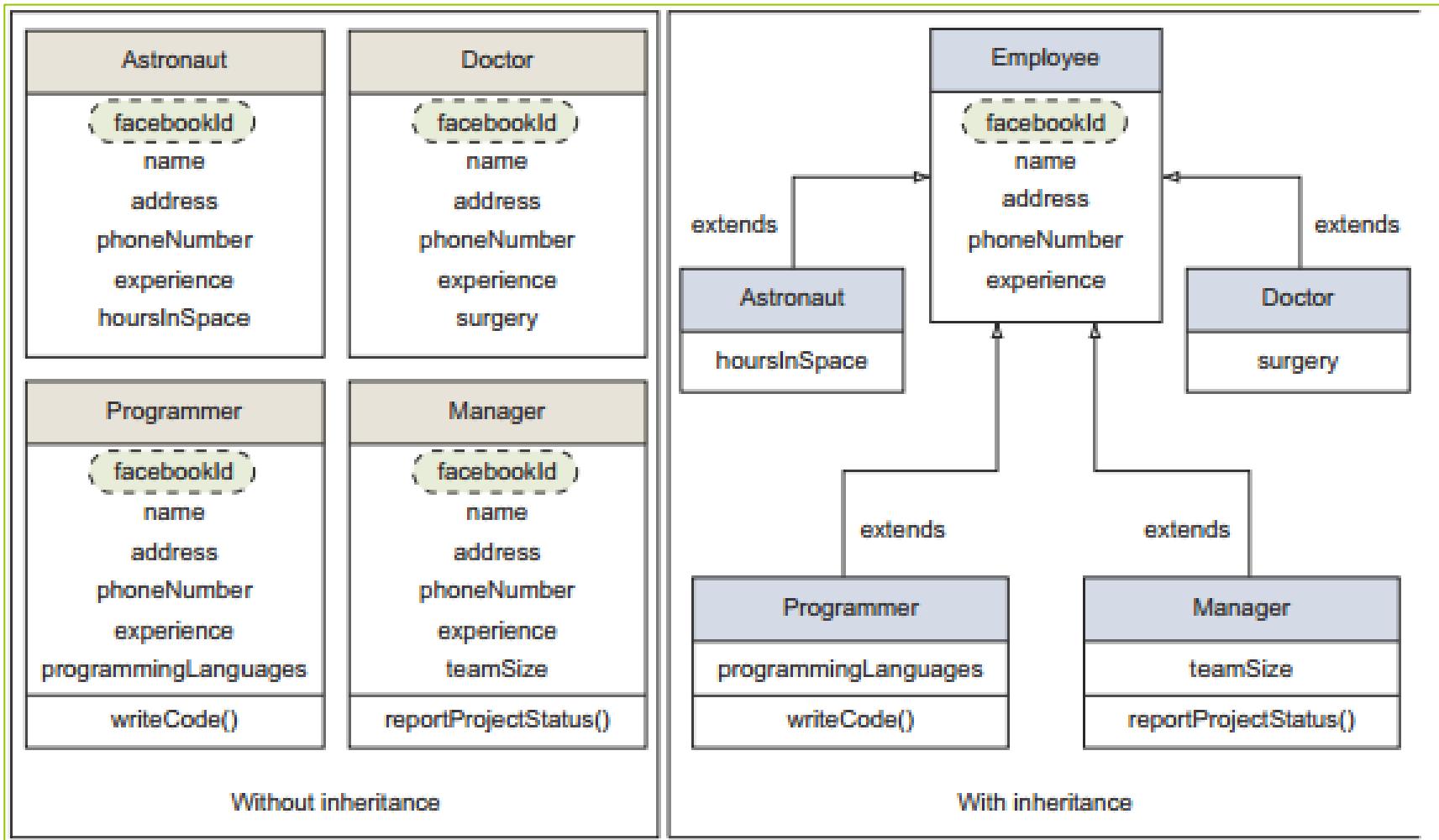
Paveldējimas

Astronaut	Doctor
name address phoneNumber experience hoursInSpace	name address phoneNumber experience surgery
Programmer	Manager
name address phoneNumber experience programmingLanguages writeCode()	name address phoneNumber experience teamSize reportProjectStatus()

Without inheritance

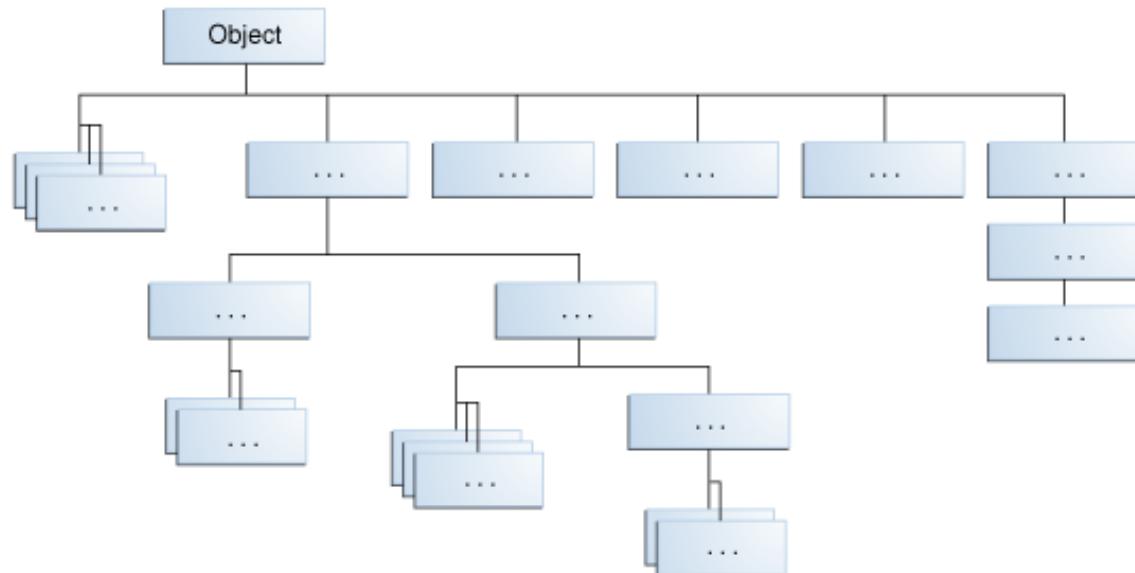


Paveldējimas



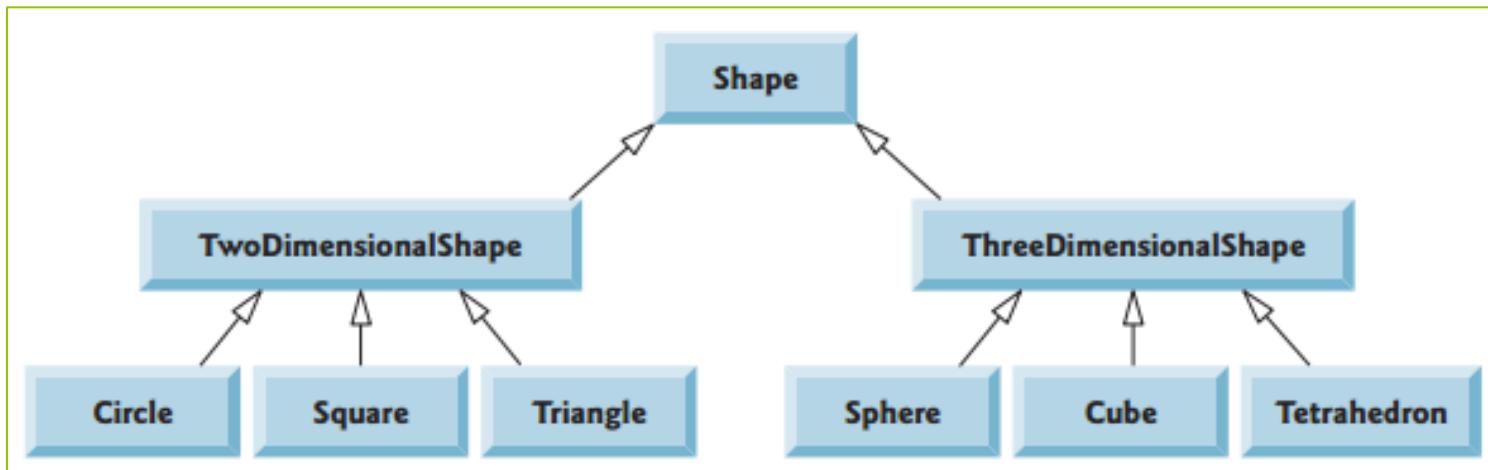
Paveldėjimas

- Klasė gali turėti tik vieną tiesioginę superklasę
- Visos Javos klasės yra kilusios iš *java.lang.Object* klasės ir automatiškai paveldi visus jos metodus



is-a relationship

Superclass	Subclasses
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
BankAccount	CheckingAccount, SavingsAccount



Machine ir Car klasės | Pavyzdys (1)

```
public class Machine {  
  
    public void start() {  
        System.out.println("Machine started.");  
    }  
  
    public void stop() {  
        System.out.println("Machine stopped.");  
    }  
}
```

Machine.java

```
public class Car extends Machine {  
  
}
```

Car.java

Main klasē | Pavyzdys (1.2)

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Machine m1 = new Machine();  
  
        System.out.println("Machine class: ");  
        m1.start();  
        m1.stop();  
  
        Car m2 = new Car();  
        // Machine m2 = new Car();  
  
        System.out.println("Car class: ");  
        m2.start();  
        m2.stop();  
  
    }  
}
```

Main.java

Machine class:
Machine started.
Machine stopped.
Car class:
Machine started.
Machine stopped.

Papildome Car klasē | Pavyzdys (2)

```
public class Car extends Machine {  
  
    public void turnRight() {  
        System.out.println("Car turned right.");  
    }  
  
    public void turnLeft() {  
        System.out.println("Car turned left.");  
    }  
}
```

Car.java

Vaikinė klasė papildoma jai
trūkstamais metodais.

Main klasė | Pavyzdys (2.1)

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Car m2 = new Car();  
  
        System.out.println("Car class: ");  
  
        m2.start();  
        m2.stop();  
  
        m2.turnLeft();  
        m2.turnRight();  
  
    }  
}
```

Car class:
Machine started.
Machine stopped.
Car turned left.
Car turned right.

Main.java

Metodų užklotis / Overriding (1)

- Vaiko klasėje paveldėtus Tėvo klasės metodus galima keisti.
- Turi sutapti:
 - metodų vardai
 - jų antraštės
 - metodų grąžinamos reikšmės tipai (gali būti labiau specifinis tipas)
- Priėjimo modifikatorius išvestinėje klasėje neturi padidinti metodo uždarumo laipsnio
- Užklojantysis metodas neturi mesti naujų *checked exceptions*
- Neužklojamai
 - konstruktoriai
 - *final* tipo metodai
 - statiniai metodai (bet galima „paslėpti“ statiniu metodu)

Machine ir Car klasēs | Pavyzdys (3)

```
public class Machine {  
  
    public void start() {  
        System.out.println  
            ("Machine started.");  
    }  
  
    public void stop() {  
        System.out.println  
            ("Machine stopped.");  
    }  
}
```

Machine.java

```
public class Car extends Machine {  
  
    @Override  
    public void start() {  
        System.out.println  
            ("Car started.");  
    }  
  
    @Override  
    public void stop() {  
        System.out.println  
            ("Car stopped.");  
    }  
  
    public void turnRight() {}  
  
    public void turnLeft() {}  
}
```

Car.java

Main klasė | Pavyzdys (3.1)

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Machine m1 = new Machine();  
  
        System.out.println("Machine class: ");  
        m1.start();  
        m1.stop();  
  
        Car m2 = new Car();  
  
        System.out.println("Car class: ");  
        m2.start();  
        m2.stop();  
  
    }  
}
```

Main.java

```
Machine class:  
Machine started.  
Machine stopped.  
Car class:  
Car started.  
Car stopped.
```

@Override – anotacija

- Norint pabrėžti, kad metodas yra perrašytas galima nurodyti anotaciją **@Override**, nors tai nėra būtina
- Taip užtikrinama kontrolė, kad metodas perrašytas, o ne naujai sukurtas

The screenshot shows an IDE interface with three tabs at the top: BaseClass.java, ChildClass.java, and OverrideTest.java. The ChildClass.java tab is active, displaying the following code:

```
1 package com.journaldev.annotations;
2
3 public class ChildClass extends BaseClass{
4
5     @Override
6     public void doSomething(String str){
7         System.out.println("Child impl:"+str);
8     }
9
10 }
```

A red error icon is visible next to the first line of the code. Below the code editor, the IDE's navigation bar includes: Problems, Javadoc, Declaration, Console, and Call Hierarchy. The Problems tab is selected, showing the message: "1 error, 24 warnings, 0 others". Under the Description section, there is a list of errors. The first error is expanded, showing the message: "The method doSomething(String) of type ChildClass must override or implement a supertype method".

super

- Perrašančio metodo viduje yra galimybė kvesti perrašomą metodą naudojant žodelį **super**.

```
public class Superclass {  
  
    public void printMethod() {  
        System.out.println("Printed in Superclass.");  
    }  
}
```

```
public class Subclass extends Superclass {  
  
    // overrides printMethod in Superclass  
    public void printMethod() {  
        super.printMethod();  
        System.out.println("Printed in Subclass");  
    }  
    public static void main(String[] args) {  
        Subclass s = new Subclass();  
        s.printMethod();  
    }  
}
```

Printed in Superclass.
Printed in Subclass

super

- Norint iškvesti tam tikrą tėvinės klasės konstruktorių, naudojamas žodelis *super(...)*.
- Šis sakinys turi būti pats pirmasis sakinys konstruktoriuje.
- Jei toks sakinys nėra nurodytas, tuomet java kompiliatorius automatiškai jį priskiria - kviečiamas konstruktorius be argumentų.

super

```
class Employee {  
    String name;  
    String address;  
  
    Employee(String name, String address) {  
        this.name = name;  
        this.address = address;  
    }  
}  
  
class Programmer extends Employee {  
    String progLanguage;  
  
    Programmer(String name, String address, String progLang) {  
        super(name, address);  
        this.progLanguage = progLang;  
    }  
}
```

super

```
public class A {
    public A() {
        System.out.println("A's constructor called");
    }
}
public class B extends A {
    public B() {
        System.out.println("B's constructor called");
    }
}
public class C extends B {
    public C() {
        System.out.println("C's constructor called");
    }
}
public static void main(String[] args) {
    new C();
}
```

super

```
public class A {
    public A() {
        super();
        System.out.println("A's constructor called");
    }
}
public class B extends A {
    public B() {
        super();
        System.out.println("B's constructor called");
    }
}
public class C extends B {
    public C() {
        super();
        System.out.println("C's constructor called");
    }
}
...
public static void main(String[] args) {
    new C();
}
```

casting

```
public class A {  
    //statements  
}  
  
public class B extends A {  
    public void foo() {}  
}  
  
A a = new B();  
  
//a.foo();  
//To execute **foo()** method.  
  
((B) a).foo();
```



VILNIAUS TECHNOLOGIJŲ IR VERSLO
PROFESINIO MOKYMO CENTRAS

10 - OBJEKTINIS PROGRAMAVIMAS 3

Jaroslav Grablevski / Justina Balsė

Casting

```
class Animal {
    void makeNoise() {
        System.out.println("generic noise");
    }
}
class Dog extends Animal {
    void makeNoise() {
        System.out.println("bark");
    }
    void playDead() {
        System.out.println("roll over");
    }
}
class CastTest2 {
    public static void main(String[] args) {
        Animal[] a = { new Animal(), new Dog(), new Animal() };
        for (Animal animal : a) {
            animal.makeNoise();
            if (animal instanceof Dog) {
                animal.playDead(); // <- won't compile
                Dog d = (Dog) animal; // casting the reference variable
                d.playDead(); // sometimes called a downcast, because we're casting
                               // down the inheritance tree to a more specific class
            }
        }
    }
}
```

Casting

Compiles but fails later:

```
class Animal {}  
class Dog extends Animal {}  
class DogTest {  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        Dog d = (Dog) animal; // compiles but fails later  
    }  
}
```

Will NOT compile :

```
public class Pzv {  
    Animal animal = new Animal();  
    Dog d = (Dog) animal;  
    String s = (String) animal; // animal can't EVER be a String  
}
```

instanceof

The **instanceof** operator tests if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular interface.

```
class Animal { }
class Dog extends Animal { }
```

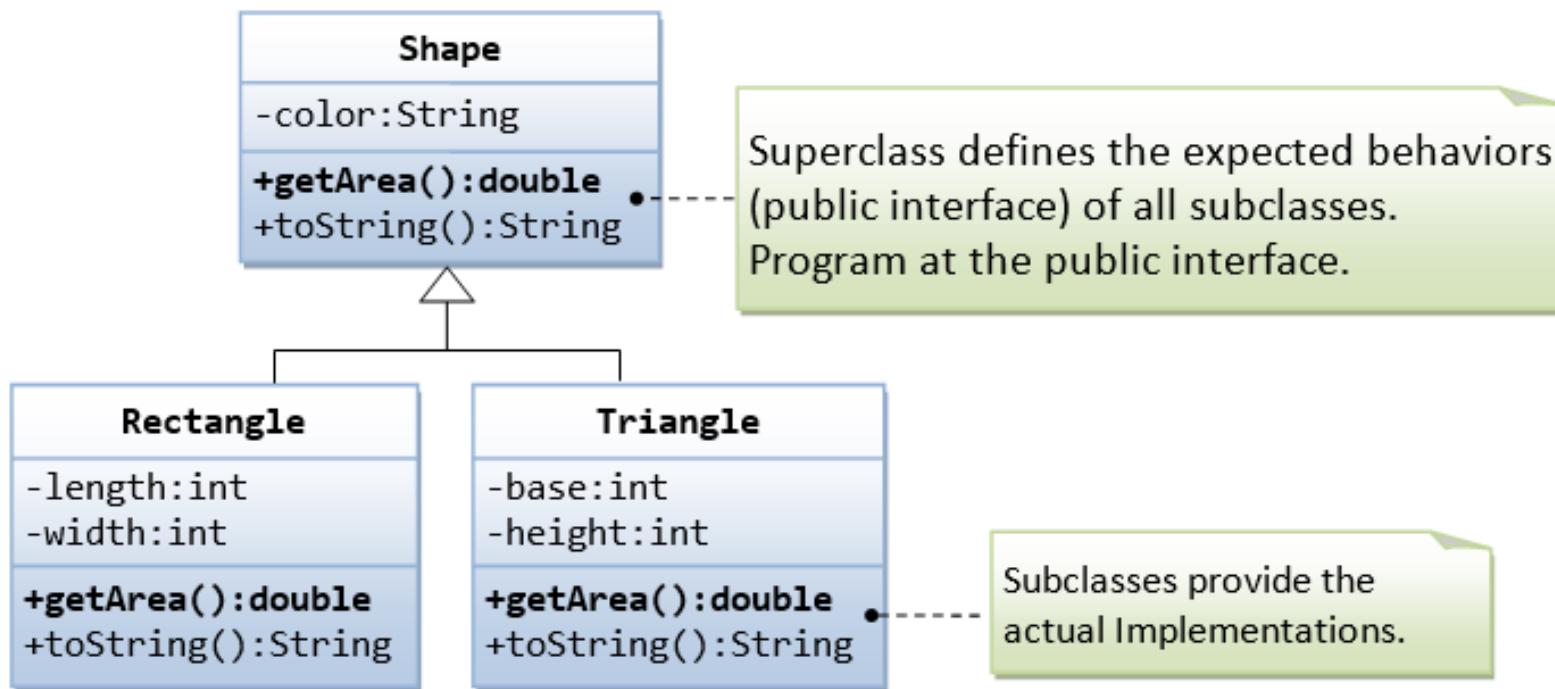
```
Dog dog = new Dog();
Animal animal = new Animal();
System.out.println(dog instanceof Dog);      //true
System.out.println(dog instanceof Animal);    //true
System.out.println(dog instanceof Object);    //true

System.out.println(animal instanceof Dog);    //false
dog = null;
System.out.println(dog instanceof Dog);        //false
```

Polimorfizmas

- objektiniame programavime naudojama sąvoka, kai metodas gali būti vykdomas skirtingai, priklausomai nuo konkrečios klasės realizacijos, metodo kvietėjui nieko nežinant apie tokius skirtumus.
- “moku dirbt su vienu tipu, moku dirbt su visais tipais, kurie įgyvendina tą patį kontraktą”
 - Grįstas paveldėjimu ir įgyvendinimu*
 - Jei Y extends X , Y galime naudoti visur kur tikimes gauti X

Polimorfizmas



Polimorfizmas pvz.

```
public class Shape {  
    // All shapes must have a method called getArea().  
    public double getArea() {  
        System.err.println("Shape unknown! Cannot compute area!");  
        return 0; // We need to return some value to compile the program.  
    }  
}  
  
public class Rectangle extends Shape {  
    private int length, width;  
    public Rectangle(int length, int width) {}  
    @Override // Override to provide the proper implementation  
    public double getArea() {  
        return length * width;  
    }  
}  
  
public class Triangle extends Shape {  
    //...  
    public double getArea() {  
        return 0.5 * base * height;  
    }  
    //...
```

Polimorfizmas p̄vz.

```
public class TestShape {  
    public static void main(String[] args) {  
        Shape s1 = new Rectangle(4, 5); // Upcast  
        System.out.println("Area is " + s1.getArea()); // Runs Rectangle's getArea()  
  
        Shape s2 = new Triangle(4, 5); // Upcast  
        System.out.println("Area is " + s2.getArea()); // Runs Triangle's getArea()  
    }  
}
```

Area is 20.0

Area is 10.0

Polimorfizmas puz. 2

```
public class Animal {  
    public void eat() {  
        System.out.println("Generic Animal Eating Generically");  
    }  
}
```

```
public class Horse extends Animal {  
    public void eat() {  
        System.out.println("Horse eating hay ");  
    }  
  
    public void eat(String s) {  
        System.out.println("Horse eating " + s);  
    }  
}
```

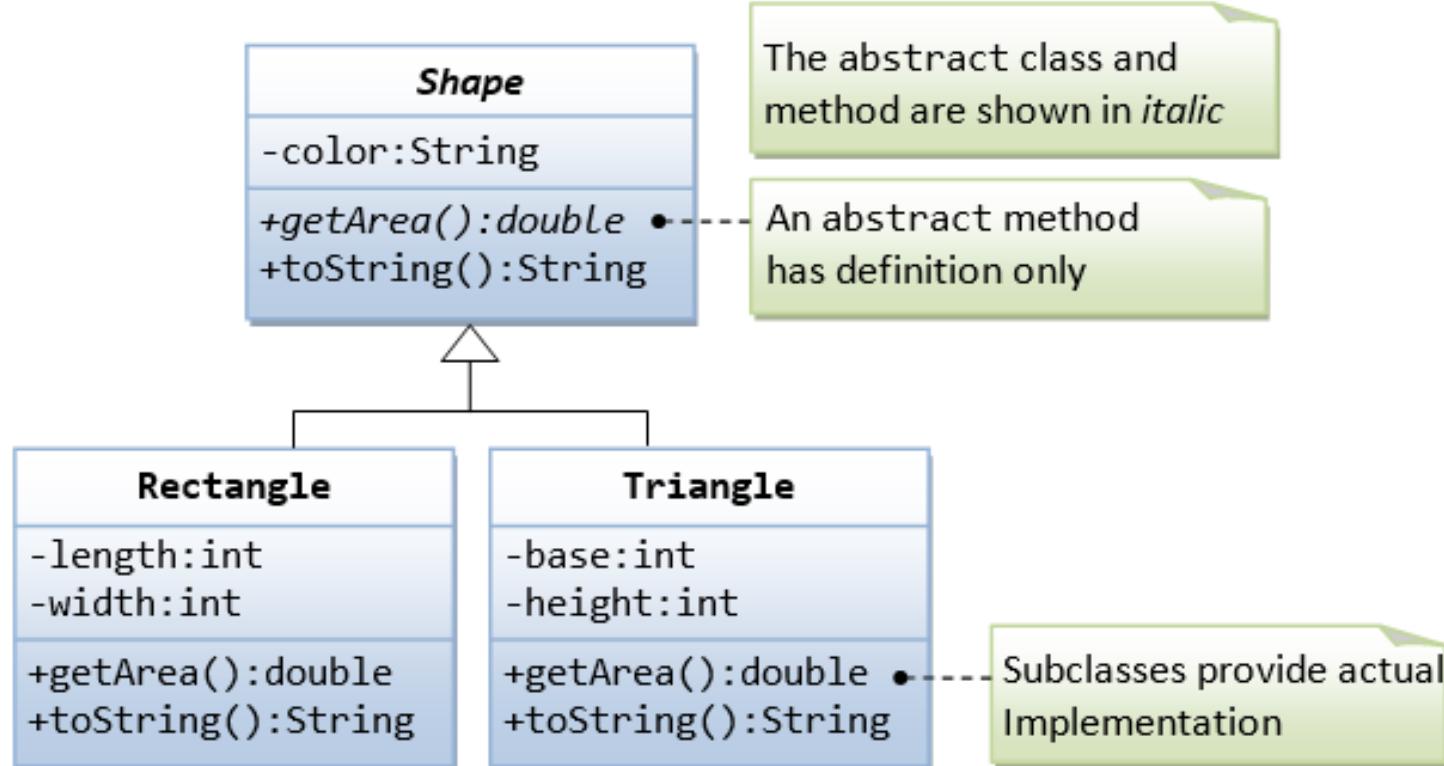
Polimorfizmas pvz. 2

Method Invocation Code	Result
<code>Animal a = new Animal(); a.eat();</code>	<code>Generic Animal Eating Generically</code>
<code>Horse h = new Horse(); h.eat();</code>	<code>Horse eating hay</code>
<code>Animal ah = new Horse(); ah.eat();</code>	<code>Horse eating hay</code> Polymorphism works—the actual object type (<code>Horse</code>), not the reference type (<code>Animal</code>), is used to determine which <code>eat()</code> is called.
<code>Horse he = new Horse(); he.eat("Apples");</code>	<code>Horse eating Apples</code> The overloaded <code>eat(String s)</code> method is invoked.
<code>Animal a2 = new Animal(); a2.eat("treats");</code>	Compiler error! Compiler sees that the <code>Animal</code> class doesn't have an <code>eat()</code> method that takes a <code>String</code> .
<code>Animal ah2 = new Horse(); ah2.eat("Carrots");</code>	Compiler error! Compiler still looks only at the reference and sees that <code>Animal</code> doesn't have an <code>eat()</code> method that takes a <code>String</code> . Compiler doesn't care that the actual object might be a <code>Horse</code> at runtime.

Abstrakti klasė

- Abstrakčios klasės turi būti pažymėtos raktažodžiu *abstract*;
- Negalima sukurti objektų iš abstrakčių klasių;
- Gali turėti nerealizuotų/abstrakčių metodų;
- Abstrakti klasė skirta paveldėjimui - paveldinti klasė turi realizuoti visus abstrakčius metodus arba pati būti abstrakčia;

Abstrakti klase



Abstrakti klase

```
abstract public class Shape {  
    // Private member variable  
    private String color;  
  
    // All Shape subclasses must implement a method called getArea()  
    abstract public double getArea();  
  
    // Constructor  
    public Shape(String color) {  
        this.color = color;  
    }  
  
    @Override  
    public String toString() {  
        return "Shape [color=" + color + "]";  
    }  
}
```

```
Shape s3 = new Shape("green"); // Compilation Error!!
```

Interfeisas

- Tai yra visiškai abstrakti klasė, kurioje egzistuoja grupė susijusių metodų, kurie yra nerealizuoti*.
- Naudojama nusakyti kontraktą
 - “Aš esu ...”
 - “Aš moku ...”
- Klasė gali įgyvendinti daug interfeisų

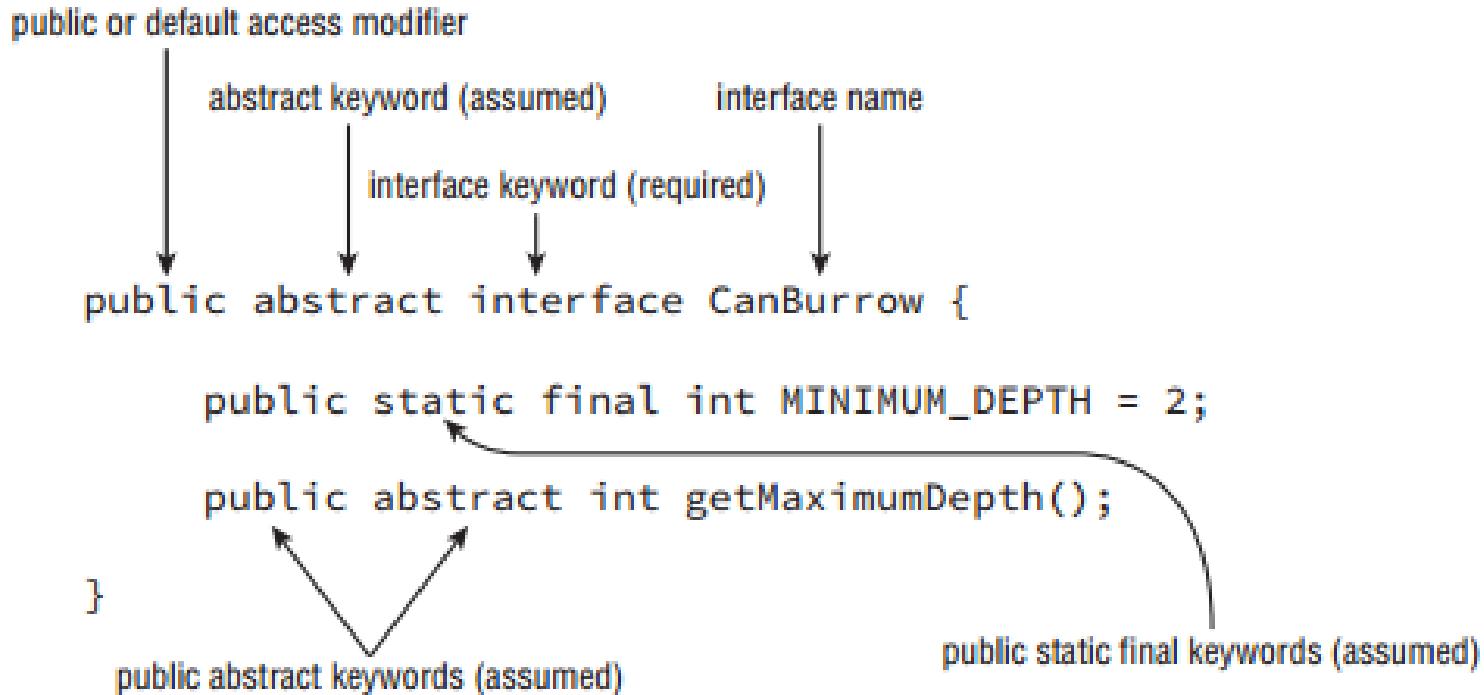
*Nuo Java 8 gali tureti realizuotus metodus (*default* ir *static*)

Interface vs Abstract class

	Java 7 and Earlier	Java 8 and Later
Abstract Classes	<ul style="list-style-type: none">• Can have concrete methods and abstract methods• Can have static methods• Can have instance variables• Class can directly extend one	(Same as Java 7)
Interfaces	<ul style="list-style-type: none">• Can only have abstract methods – no concrete methods• Cannot have static methods• Cannot have mutable instance variables• Class can implement any number	<ul style="list-style-type: none">• Can have concrete (default) methods and abstract methods• Can have static methods• Cannot have mutable instance variables• Class can implement any number

Interfeisas

- Defining an interface



Interfeisas

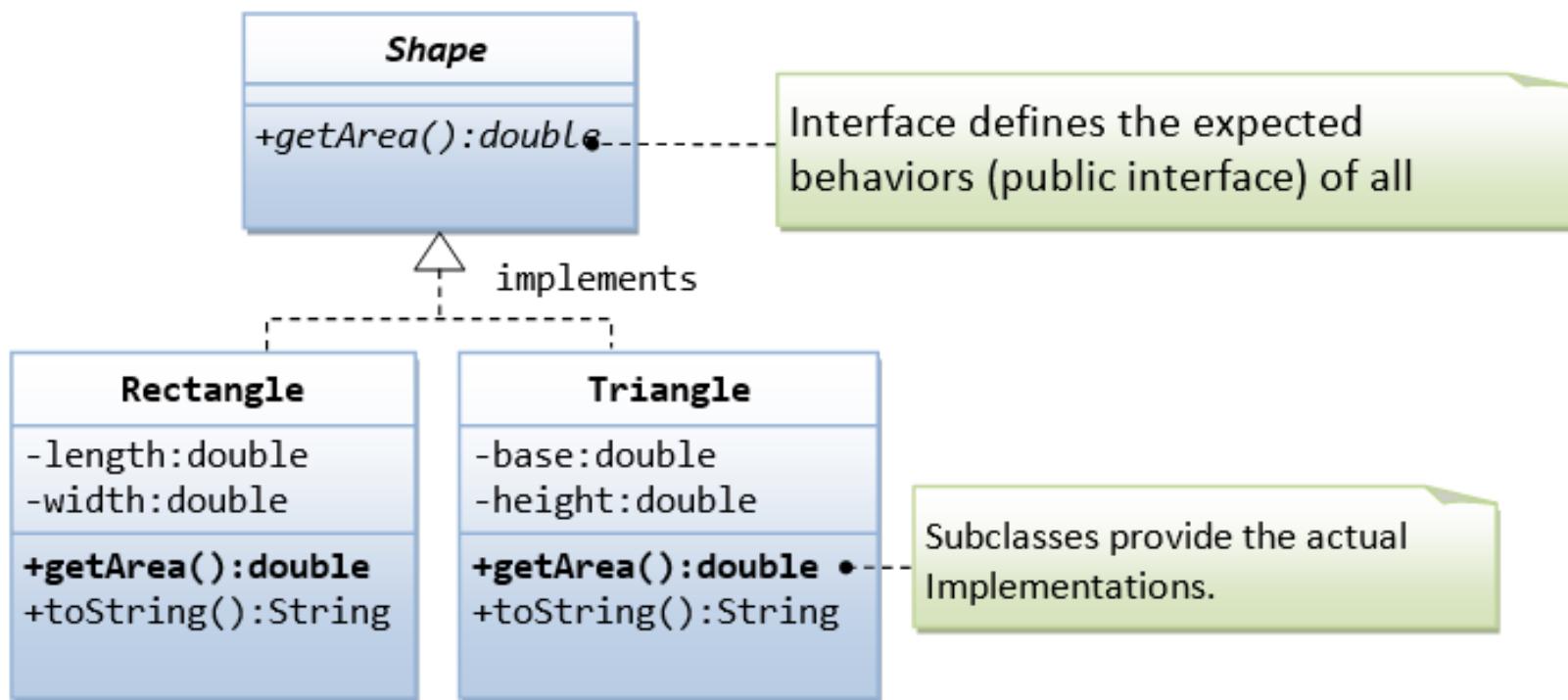
- Implementing an interface

The diagram illustrates the syntax for implementing an interface in Java. It shows a class definition with annotations pointing to its components:

- implements keyword (required)**: Points to the word "implements" in the line "implements CanBurrow".
- class name**: Points to the word "FieldMouse" in the line "public class FieldMouse".
- Interface name**: Points to the word "CanBurrow" in the line "implements CanBurrow".
- signature matches interface method**: Points to the method "getMaximumDepth" in the code.

```
public class FieldMouse implements CanBurrow {  
    public int getMaximumDepth() {  
        return 10;  
    }  
}
```

Interfeisas



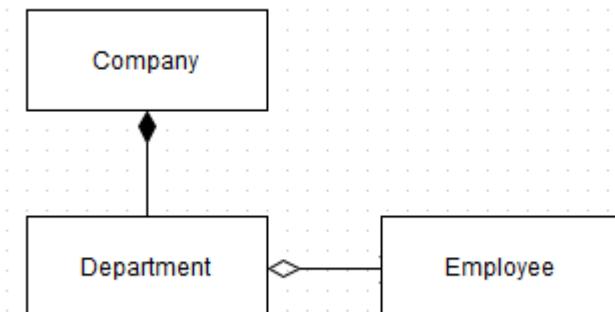
Interfeisas

```
public interface Shape { // Use keyword "interface" instead of "class"  
    // "public abstract" assumed  
    double getArea();  
}
```

```
public class Rectangle implements Shape {  
    // ...
```

Agregavimas ir kompozicija

- Agregavimo ryšys vadinamas „turi“ (angl. **has-a**) ryšiu
- Agregavimas yra silpnesnis sudėtinio objekto ryšys
 - vienos klasės objektas yra kitos objekto dalis, tačiau sunaikinus vieną, kitas nebūtinai turi būti sunaikinamas
- Kompozicijos ryšys – stipresnis
 - sunaikinus vienos klasės objektą sunaikinamos ir jo sudedamosios dalys



Agregavimas

