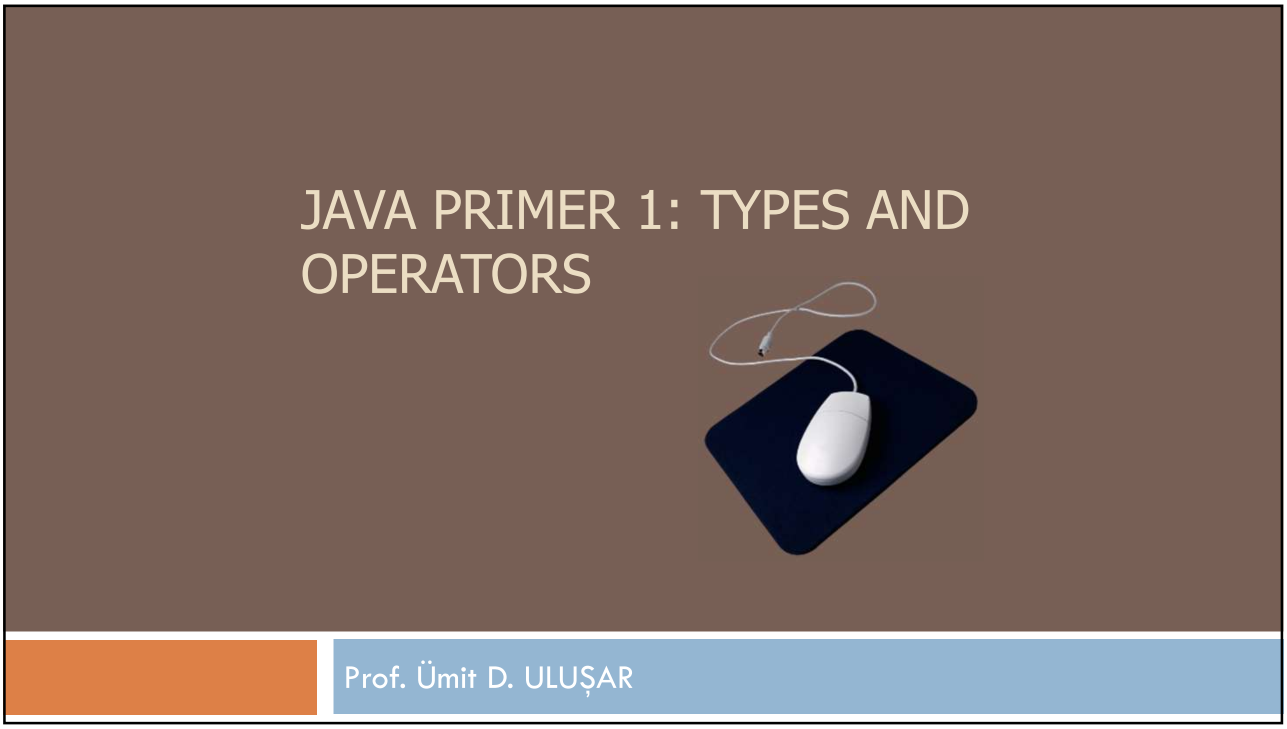
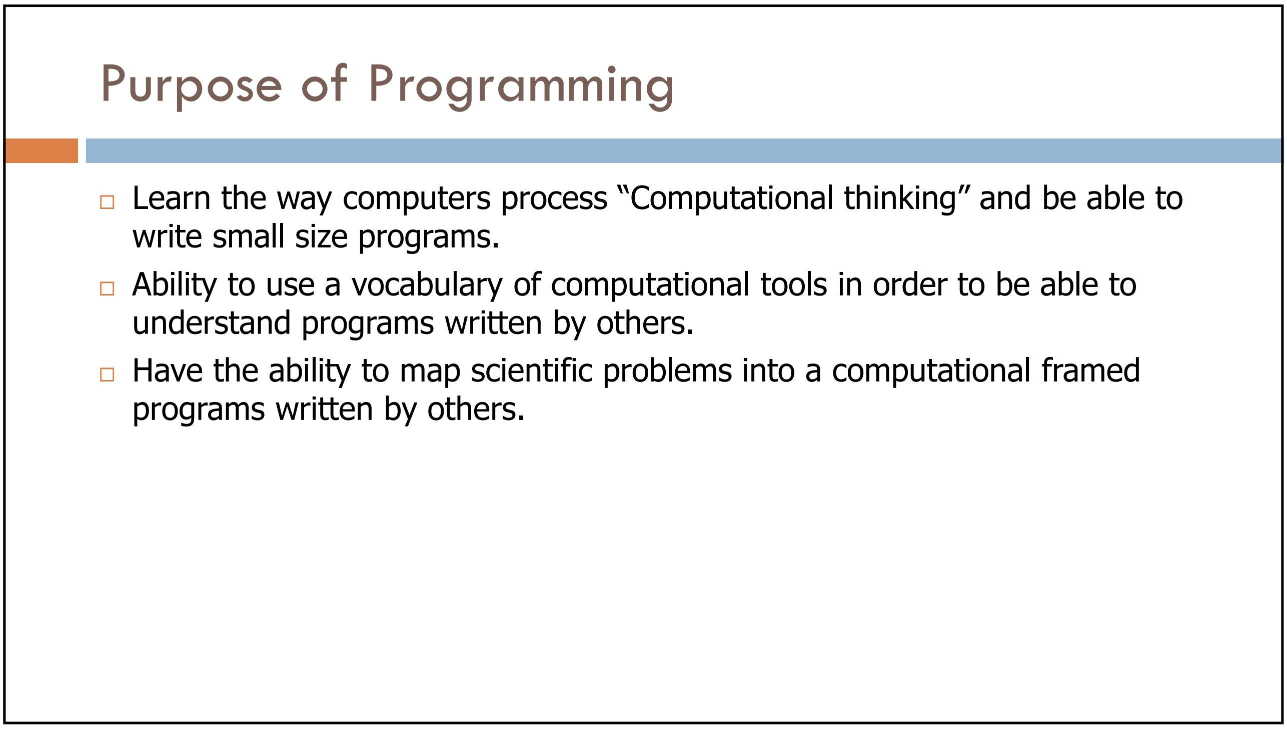


JAVA PRIMER 1: TYPES AND OPERATORS

Prof. Ümit D. ULUŞAR

1

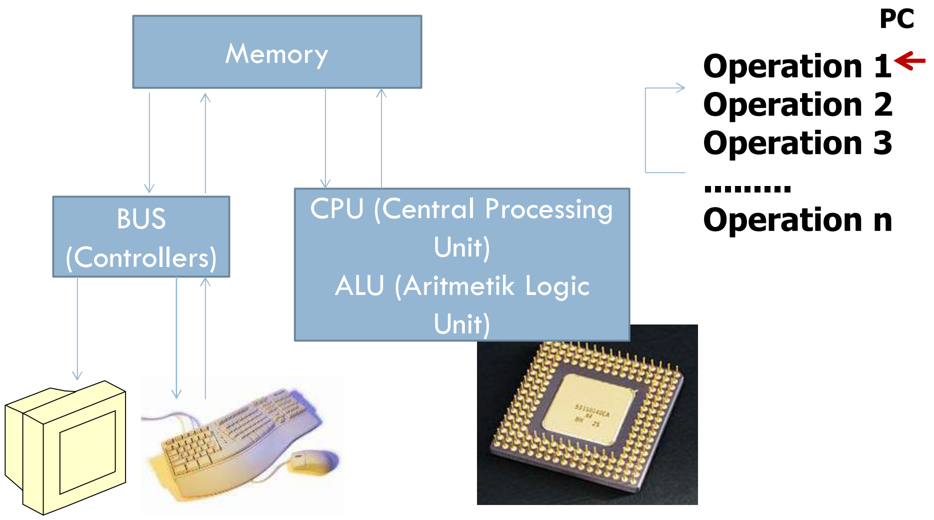


Purpose of Programming

- □ Learn the way computers process “Computational thinking” and be able to write small size programs.
- □ Ability to use a vocabulary of computational tools in order to be able to understand programs written by others.
- □ Have the ability to map scientific problems into a computational framed programs written by others.

2

Program



Programming Language

- Language is a set of valid sentences.
- What makes a language valid?
  - Syntax (Grammatical (syntactically valid))
  - Semantics (Sensible (Semantically valid))
  - Trees are walking.

Programming Languages

- Machine languages — interpreted directly in hardware
- Assembly languages — thin wrappers over a corresponding machine language
- **High-level languages — anything machine-independent**
- System languages — designed for writing low-level tasks, like memory and process management
- Scripting languages — generally extremely high-level and powerful
- Domain-specific languages — used in highly special-purpose areas only
- Visual languages — non-text based

5

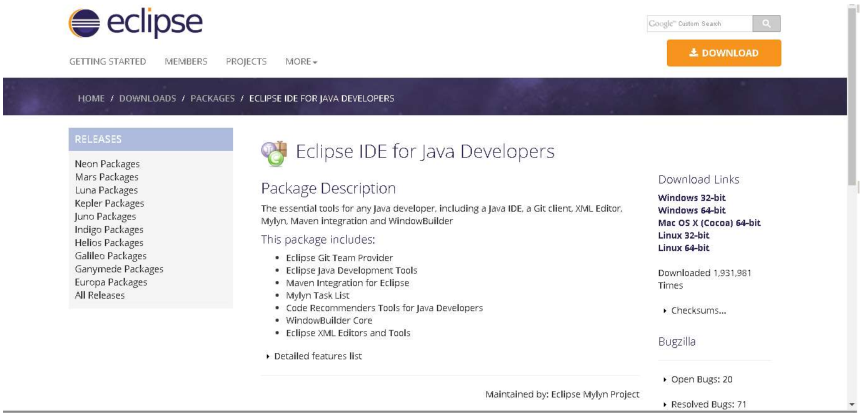
Programming Language Java

- Java Features
  - Widely used, widely available.
  - Embraces full set of modern abstraction.
  - Variety of automatic checks for mistakes in programs.
- Java Economy
  - Mars rover.
  - Cell phones (Android)
  - Web servers
  - Medical Devices
  - Super Computing

6

Java Installation (Eclipse)

http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/marsr



Eclipse IDE for Java Developers

Package Description

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven integration and WindowBuilder

This package includes:

- Eclipse Git Team Provider
- Eclipse Java Development Tools
- Maven Integration for Eclipse
- Mylyn Task List
- Code Recommenders Tools for Java Developers
- WindowBuilder Core
- Eclipse XML Editors and Tools

Downloaded 1,931,981 Times

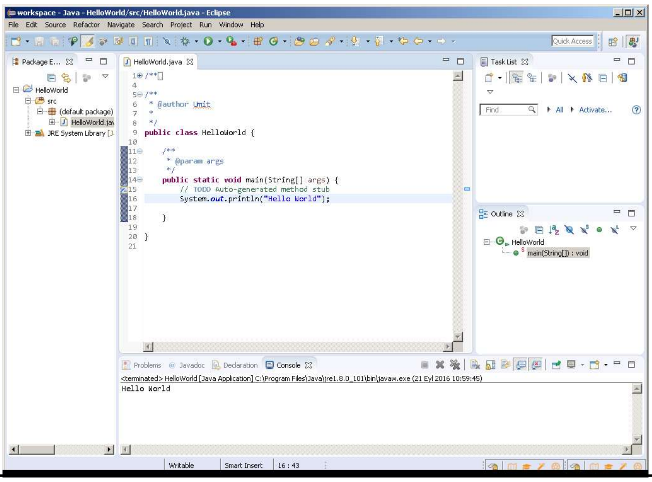
Maintained by: Eclipse Mylyn Project

The Java Compiler

- Java is a compiled language.
- Programs are compiled into byte-code executable files, which are executed through the Java virtual machine (JVM).
  - The JVM reads each instruction and executes that instruction.
- A programmer defines a Java program in advance and saves that program in a text file known as source code.
- For Java, source code is conventionally stored in a file named with the **.java** suffix (e.g., **demo.java**) and the byte-code file is stored in a file named with a **.class** suffix, which is produced by the Java compiler.

Eclipse IDE

Geting Started
http://help.eclipse.org/neon/index.jsp?nav=%2F0



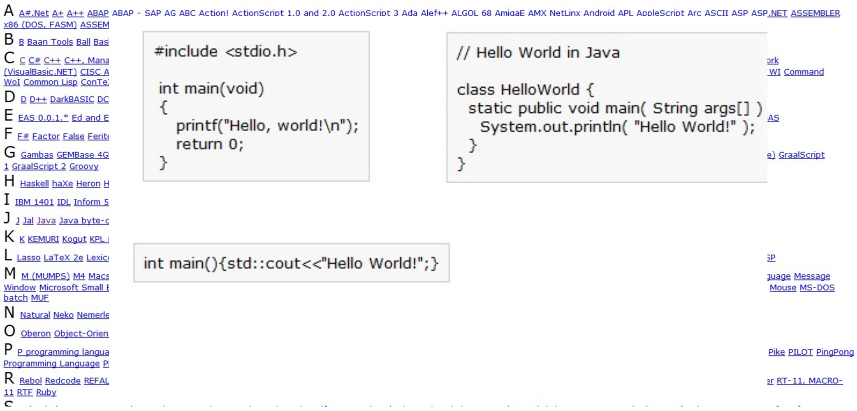
```
/**□

/**
 * @author Umit
 *
 */
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hello World");

    }

}
```

```
<terminated> HelloWorld [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (21 Eyl 2016 10:59:45)
Hello World
```

9

Hello World in Different Languages

- □ http://www.helloworldexample.net/java-hello-world-example.html

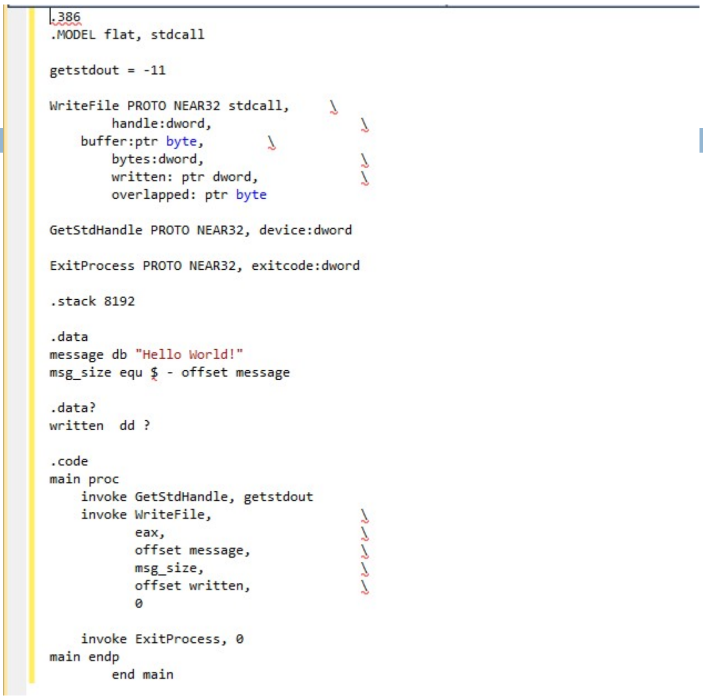


```
#include <stdio.h>

 int main(void)
 {
    printf("Hello, world!\n");
    return 0;
 }
```

```
// Hello World in Java

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

```
int main(){std::cout<<"Hello World!";}
```

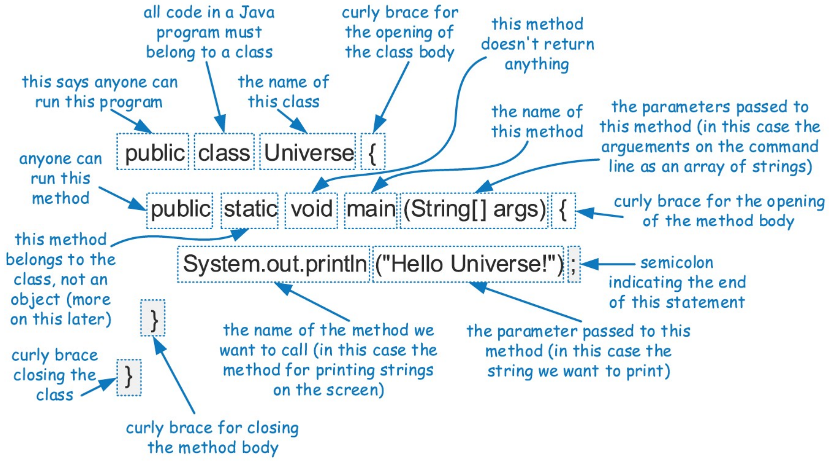
10



```
.386
.MODEL flat, stdcall

getstdout = -11

WriteFile PROTO NEAR32 stdcall,     \
        handle:dword,               \
    buffer:ptr byte,        \
        bytes:dword,                \
        written: ptr dword,         \
        overlapped: ptr byte

GetStdHandle PROTO NEAR32, device:dword

ExitProcess PROTO NEAR32, exitcode:dword

.stack 8192

.data
message db "Hello World!"
msg_size equ $ - offset message

.data?
written  dd ?

.code
main proc
    invoke GetStdHandle, getstdout
    invoke WriteFile,                \
           eax,                      \
           offset message,           \
           msg_size,                 \
           offset written,           \
           0

    invoke ExitProcess, 0
main endp
        end main
```

11

Program Structure



all code in a Java program must belong to a class

curly brace for the opening of the class body

this method doesn't return anything

this says anyone can run this program

the name of this class

the name of this method

the parameters passed to this method (in this case the arguements on the command line as an array of strings)

anyone can run this method

curly brace for the opening of the method body

this method belongs to the class, not an object (more on this later)

semicolon indicating the end of this statement

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

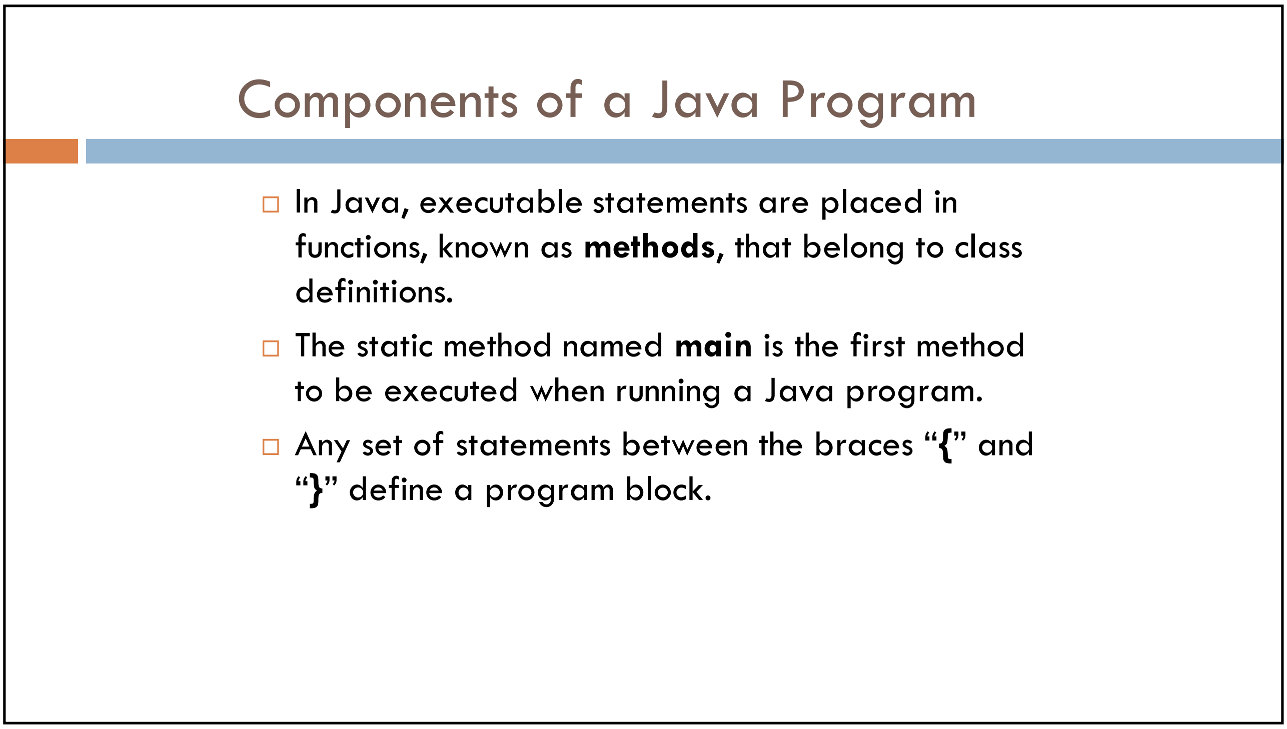
the name of the method we want to call (in this case the method for printing strings on the screen)

the parameter passed to this method (in this case the string we want to print)

curly brace closing the class

curly brace for closing the method body

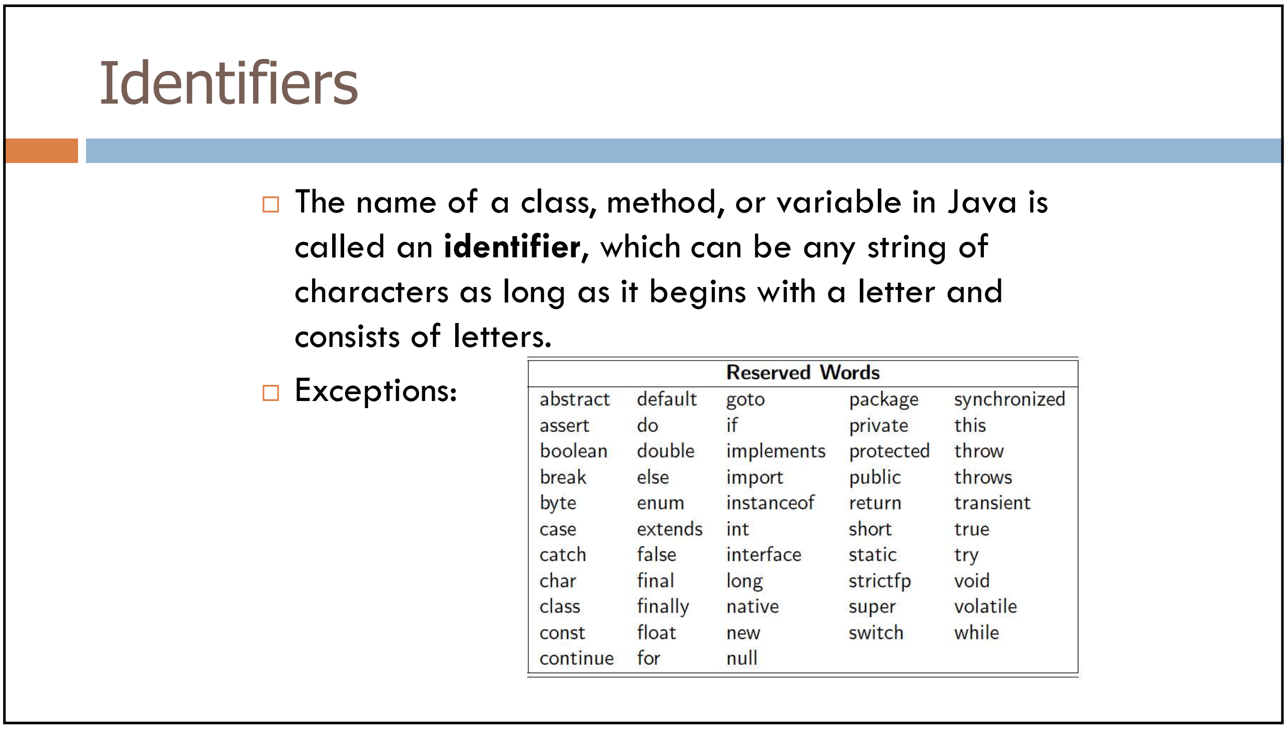
12



Components of a Java Program

- □ In Java, executable statements are placed in functions, known as **methods**, that belong to class definitions.
- □ The static method named **main** is the first method to be executed when running a Java program.
- □ Any set of statements between the braces "**{**" and "**}**" define a program block.

13



Identifiers

- □ The name of a class, method, or variable in Java is called an **identifier**, which can be any string of characters as long as it begins with a letter and consists of letters.
- □ Exceptions:

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

14

Built-in Types

- □ Java has several built-in types, which are basic ways of storing data.
- □ An identifier variable can be declared to hold any base type and it can later be reassigned to hold another value of the same type.

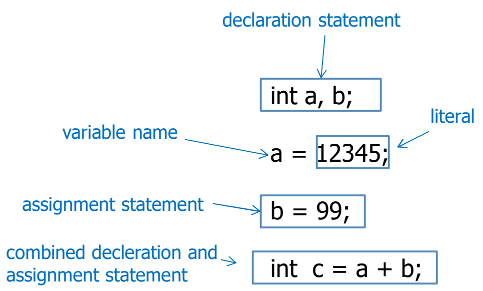
boolean	a boolean value: true or false
char	16-bit Unicode character
byte	8-bit signed two's complement integer
short	16-bit signed two's complement integer
int	32-bit signed two's complement integer
long	64-bit signed two's complement integer
float	32-bit floating-point number (IEEE 754-1985)
double	64-bit floating-point number (IEEE 754-1985)

```
boolean flag = true;
boolean verbose, debug;
char grade = 'A';
byte b = 12;
short s = 24;
int i, j, k = 257;
long l = 890L;
float pi = 3.1416F;
double e = 2.71828, a = 6.022e23;
```

15

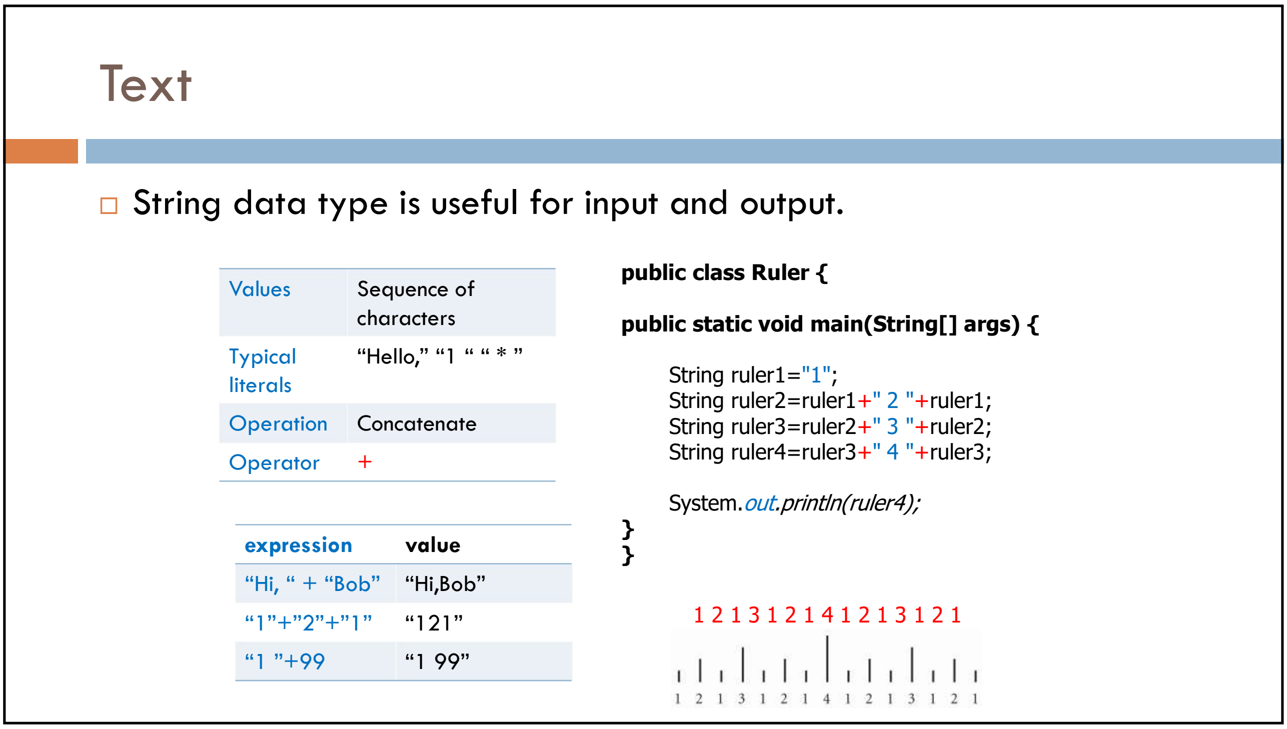
Variables

- □ Variable is a name that refers to a value.
- □ Assignment statement associates a value with a variable.



```
int a, b;
a = 12345;
b = 99;
int  c = a + b;
```

16



Text

- String data type is useful for input and output.

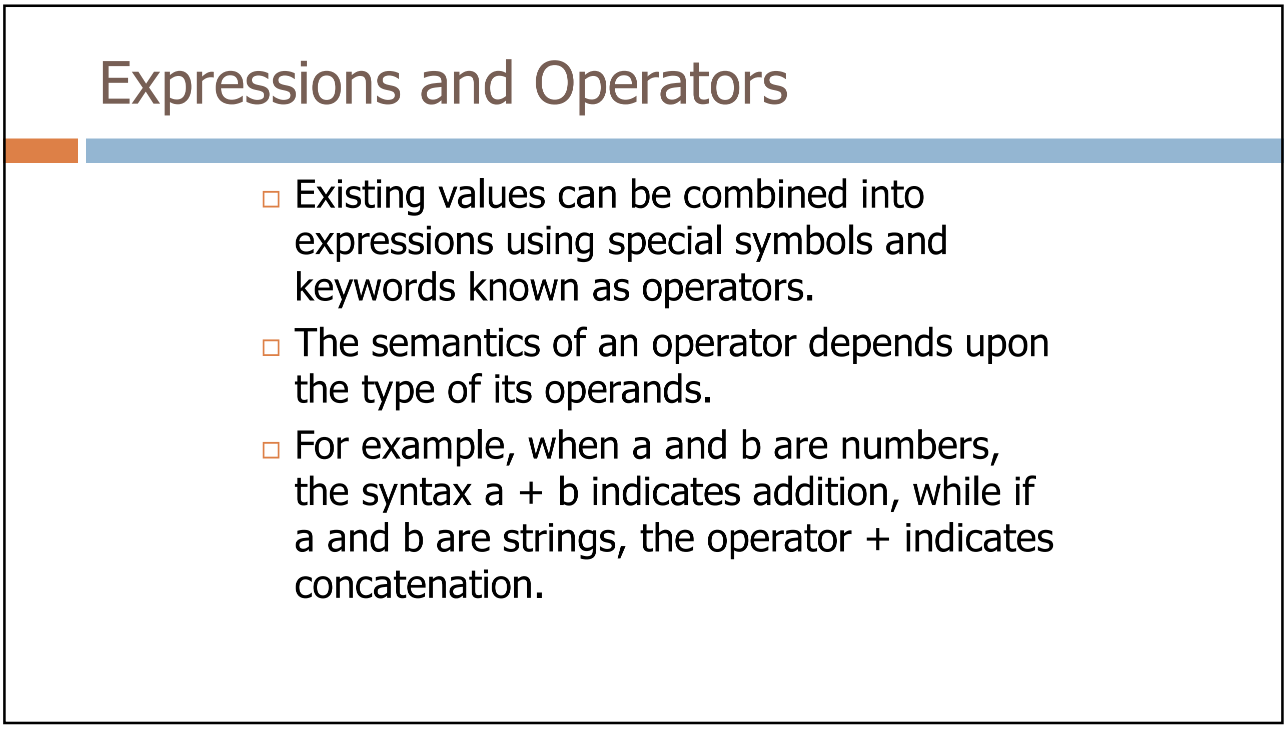
Values	Sequence of characters
Typical literals	“Hello,” “1 “ “ * ”
Operation	Concatenate
Operator	+

expression	value
“Hi, “ + “Bob”	“Hi,Bob”
“1”+”2”+”1”	“121”
“1 ”+99	“1 99”

```
public class Ruler {

public static void main(String[] args) {

      String ruler1="1";
      String ruler2=ruler1+" 2 "+ruler1;
      String ruler3=ruler2+" 3 "+ruler2;
      String ruler4=ruler3+" 4 "+ruler3;

      System.out.println(ruler4);
}
}
```

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

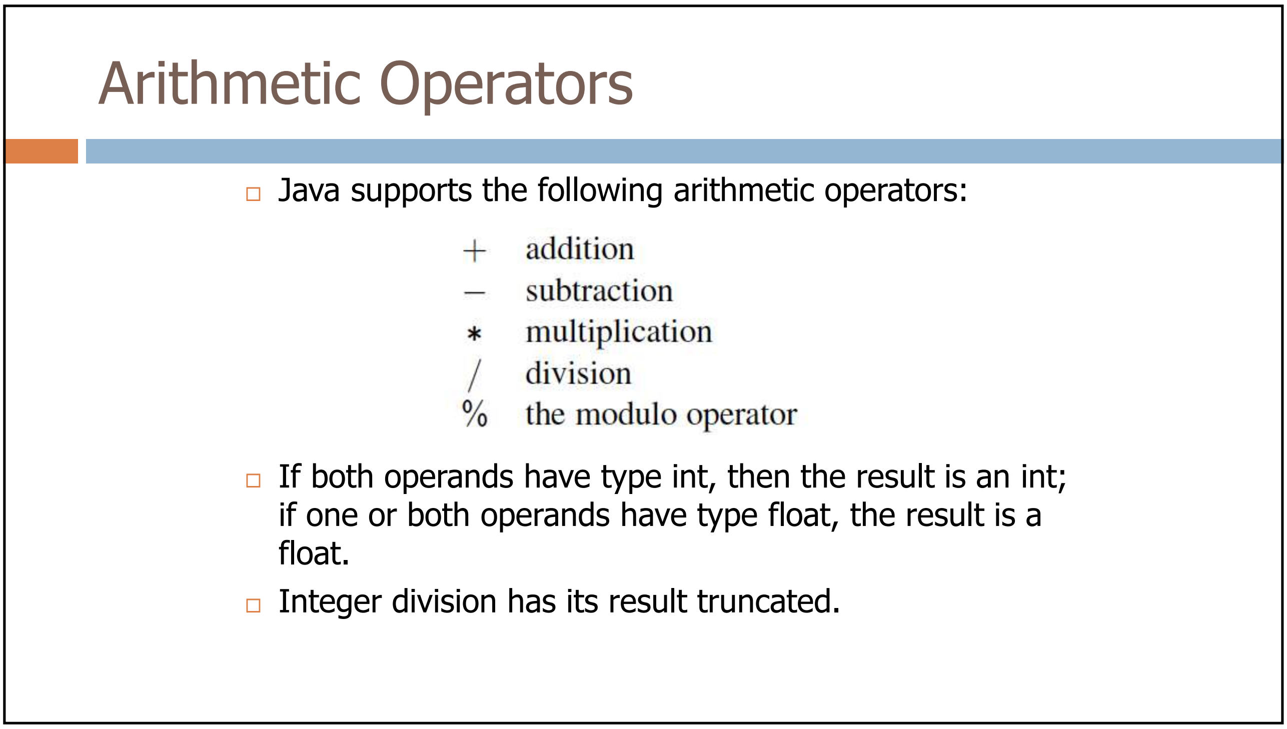
17



Expressions and Operators

- Existing values can be combined into expressions using special symbols and keywords known as operators.
- The semantics of an operator depends upon the type of its operands.
- For example, when a and b are numbers, the syntax a + b indicates addition, while if a and b are strings, the operator + indicates concatenation.

18



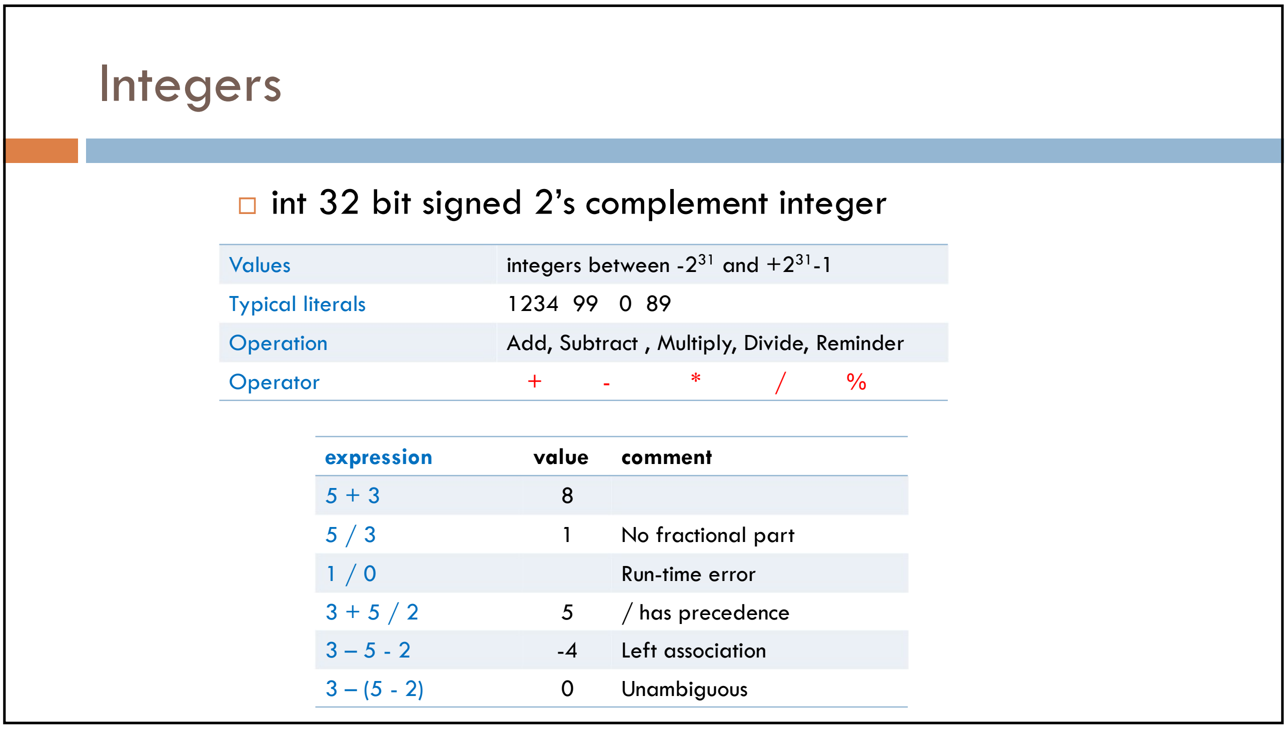
Arithmetic Operators

- Java supports the following arithmetic operators:

+	addition
−	subtraction
*	multiplication
/	division
%	the modulo operator

- If both operands have type int, then the result is an int; if one or both operands have type float, the result is a float.
- Integer division has its result truncated.

19



Integers

- int 32 bit signed 2's complement integer

Values	integers between -2^{31} and $+2^{31}-1$
Typical literals	1234 99 0 89
Operation	Add, Subtract , Multiply, Divide, Reminder
Operator	+ - * / %

expression	value	comment
5 + 3	8	
5 / 3	1	No fractional part
1 / 0		Run-time error
3 + 5 / 2	5	/ has precedence
3 – 5 - 2	-4	Left association
3 – (5 - 2)	0	Unambiguous

20

Integer Operations

```
public class IntOps {
public static void main(String[] args) {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum = a + b;
      int prod = a * b;
      int quot = a / b;
      int rem = a % b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);
      System.out.println(a + " / " + b + " = " + quot);
      System.out.println(a + " % " + b + " = " + rem);
}
}
```

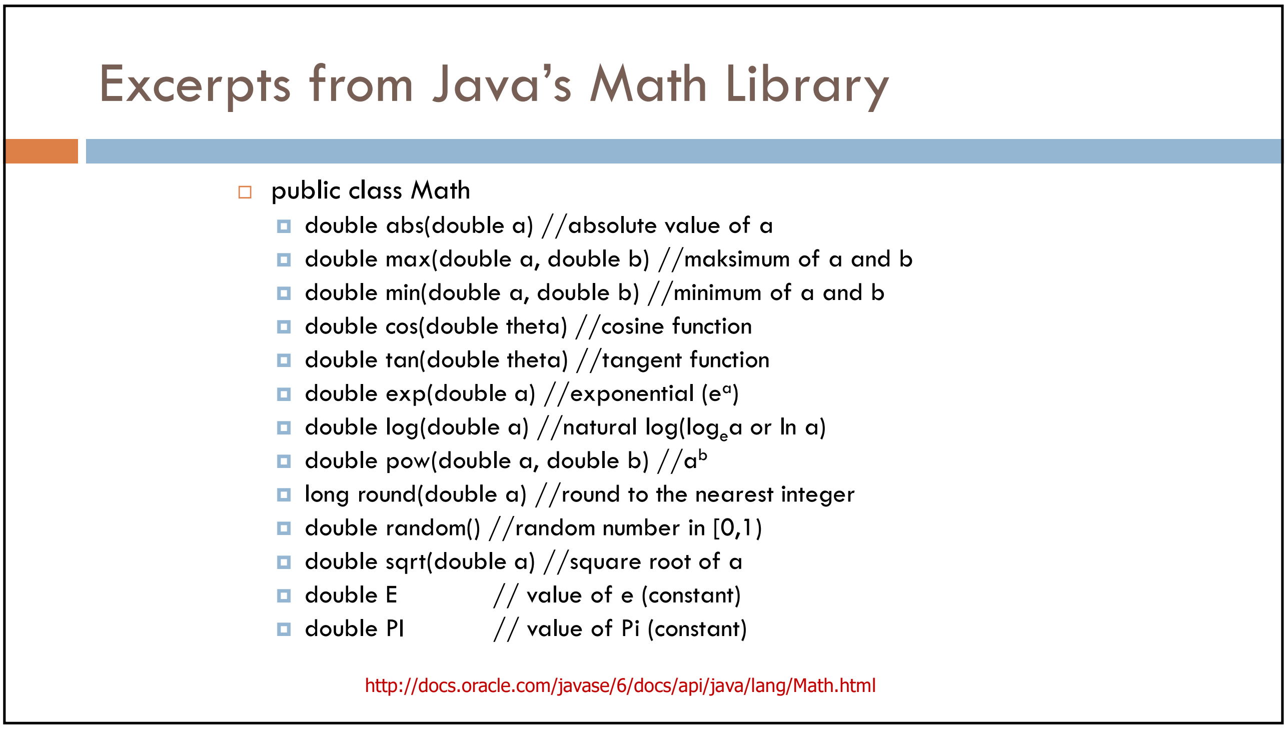
```
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46
```

Floating-Point Numbers

- double real numbers (specified by IEEE 754 standards)

Values	Real numbers between 0x0.0000000000001p-1022 and 0x1.fffffffffffffp1023
Typical literals	3.14159 6.02e23 -3.0
Operation	Add, Subtract , Multiply, Divide, Reminder
Operator	+ - * / %

expression	value
5.0 / 3.0	1.6666666666666667
6.02e23 / 2.0	3.01e23
1 / 0	Infinity
Math.sqrt(2.0)	1.4142135623730951
Math.sqrt(-1.0)	NaN

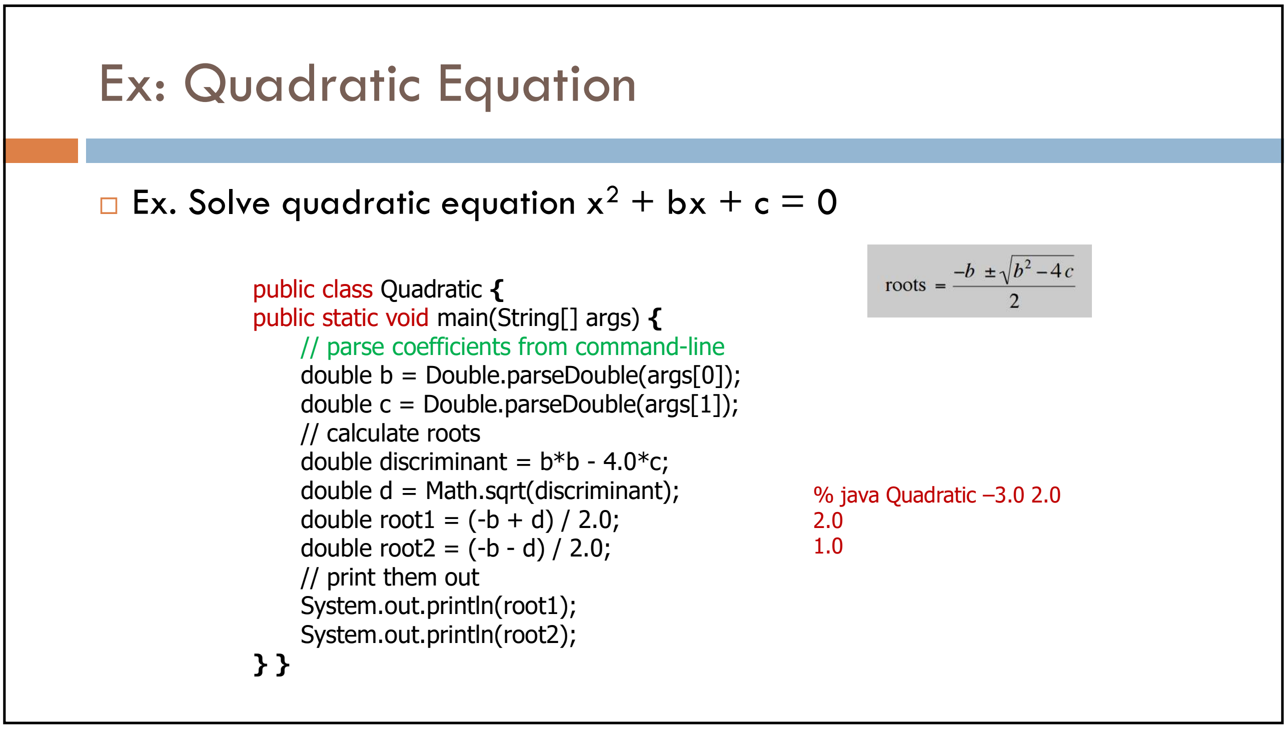


Excerpts from Java’s Math Library

- public class Math
  - double abs(double a) //absolute value of a
  - double max(double a, double b) //maksimum of a and b
  - double min(double a, double b) //minimum of a and b
  - double cos(double theta) //cosine function
  - double tan(double theta) //tangent function
  - double exp(double a) //exponential (e^a)
  - double log(double a) //natural log($\log_e$a or ln a)
  - double pow(double a, double b) //a^b
  - long round(double a) //round to the nearest integer
  - double random() //random number in [0,1)
  - double sqrt(double a) //square root of a
  - double E // value of e (constant)
  - double PI // value of Pi (constant)

http://docs.oracle.com/javase/6/docs/api/java/lang/Math.html

23



Ex: Quadratic Equation

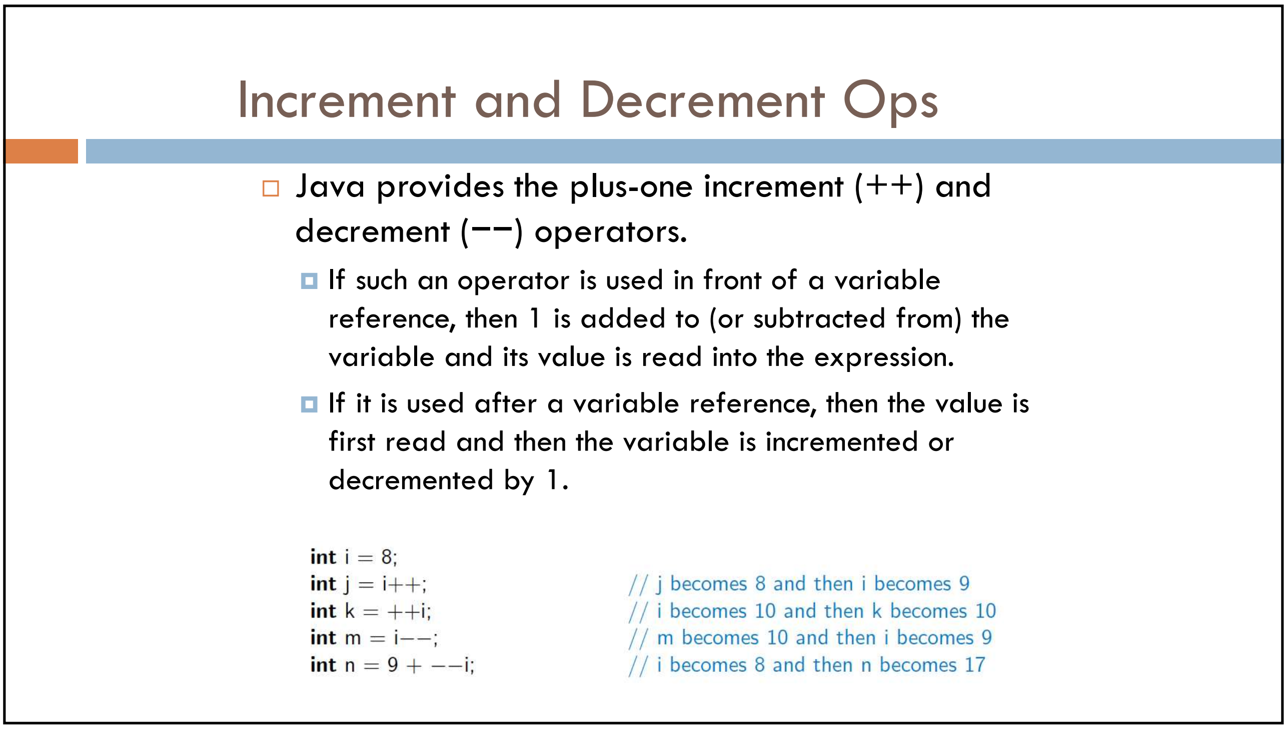
- Ex. Solve quadratic equation $x^2 + bx + c = 0$

$$\text{roots} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

```
public class Quadratic {
public static void main(String[] args) {
    // parse coefficients from command-line
    double b = Double.parseDouble(args[0]);
    double c = Double.parseDouble(args[1]);
    // calculate roots
    double discriminant = b*b - 4.0*c;
    double d = Math.sqrt(discriminant);
    double root1 = (-b + d) / 2.0;
    double root2 = (-b - d) / 2.0;
    // print them out
    System.out.println(root1);
    System.out.println(root2);
} }
```

```
% java Quadratic –3.0 2.0
2.0
1.0
```

24

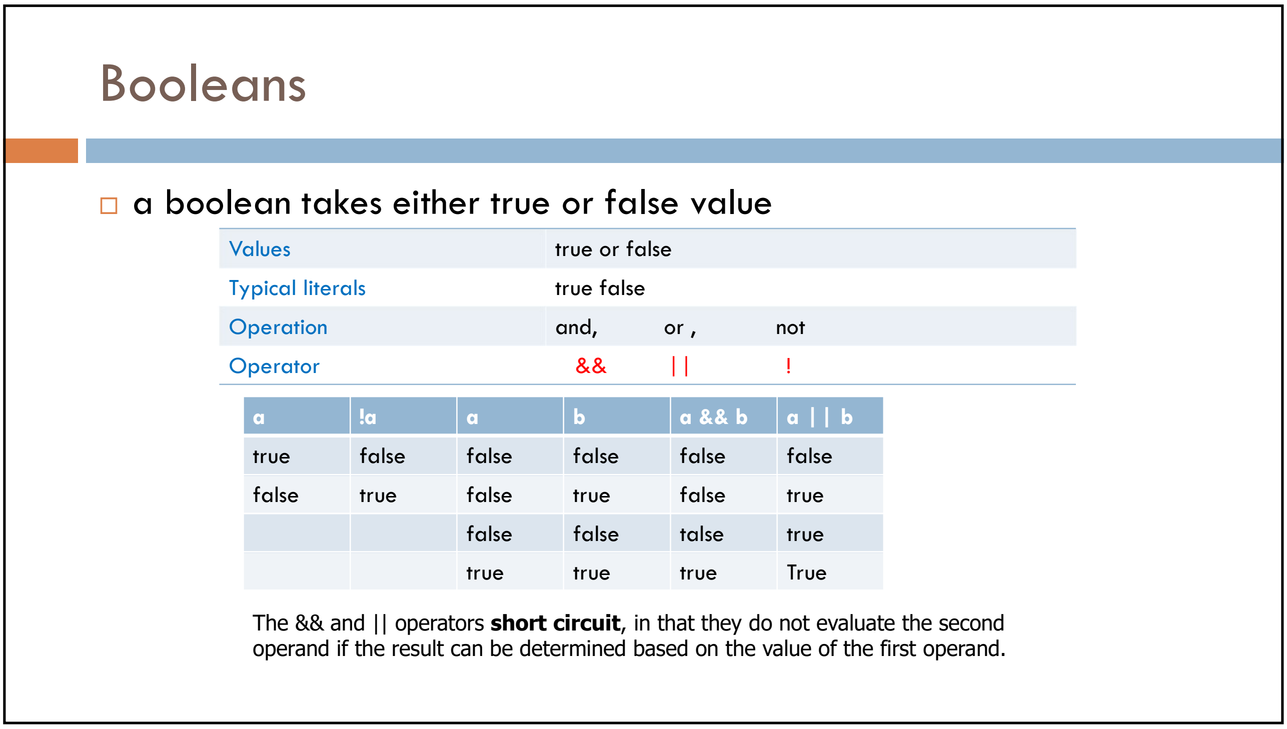


Increment and Decrement Ops

- Java provides the plus-one increment (++) and decrement (−−) operators.
  - If such an operator is used in front of a variable reference, then 1 is added to (or subtracted from) the variable and its value is read into the expression.
  - If it is used after a variable reference, then the value is first read and then the variable is incremented or decremented by 1.

```
int i = 8;
int j = i++;              // j becomes 8 and then i becomes 9
int k = ++i;              // i becomes 10 and then k becomes 10
int m = i−−;              // m becomes 10 and then i becomes 9
int n = 9 + −−i;          // i becomes 8 and then n becomes 17
```

25



Booleans

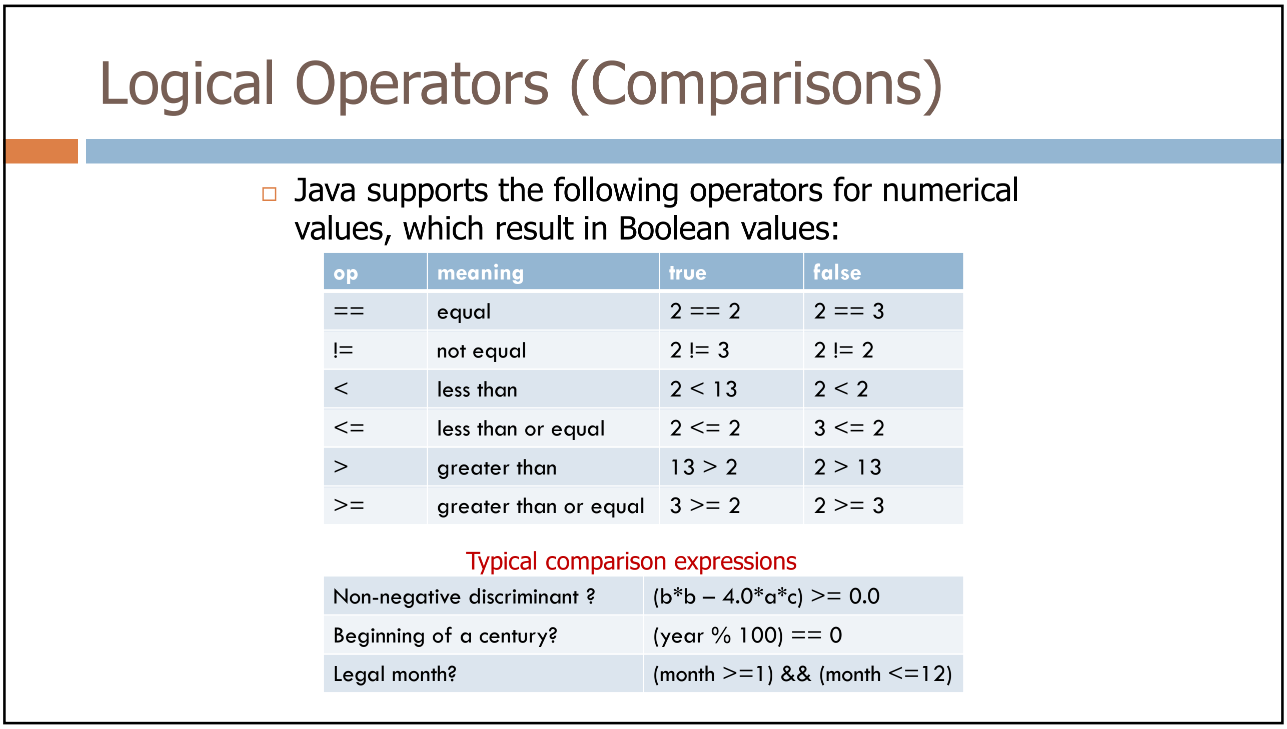
- a boolean takes either true or false value

Values	true or false		
Typical literals	true false		
Operation	and,	or ,	not
Operator	&&	||	!

a	!a	a	b	a && b	a | | b
true	false	false	false	false	false
false	true	false	true	false	true
		false	false	talse	true
		true	true	true	True

The && and || operators **short circuit**, in that they do not evaluate the second operand if the result can be determined based on the value of the first operand.

26



Logical Operators (Comparisons)

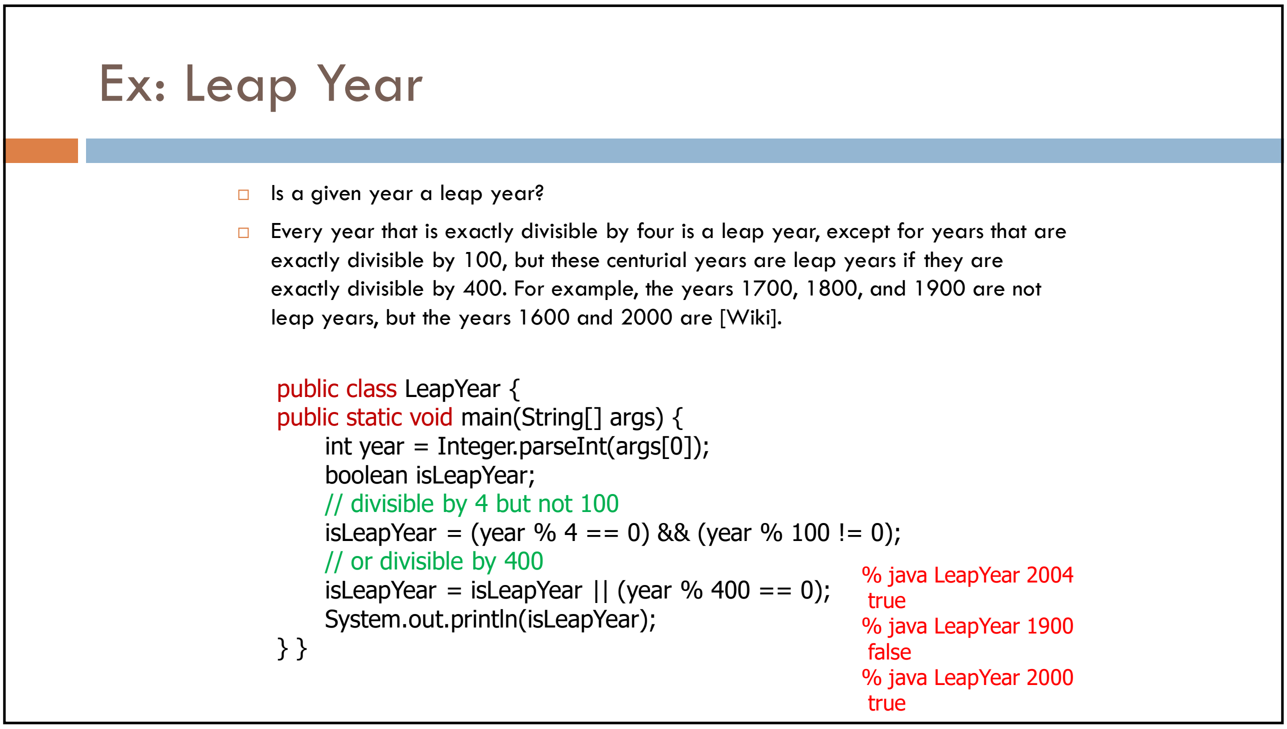
- Java supports the following operators for numerical values, which result in Boolean values:

op	meaning	true	false
==	equal	2 == 2	2 == 3
!=	not equal	2 != 3	2 != 2
<	less than	2 < 13	2 < 2
<=	less than or equal	2 <= 2	3 <= 2
>	greater than	13 > 2	2 > 13
>=	greater than or equal	3 >= 2	2 >= 3

Typical comparison expressions

Non-negative discriminant ?	(b*b – 4.0*a*c) >= 0.0
Beginning of a century?	(year % 100) == 0
Legal month?	(month >=1) && (month <=12)

27



Ex: Leap Year

- Is a given year a leap year?
- Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years if they are exactly divisible by 400. For example, the years 1700, 1800, and 1900 are not leap years, but the years 1600 and 2000 are [Wiki].

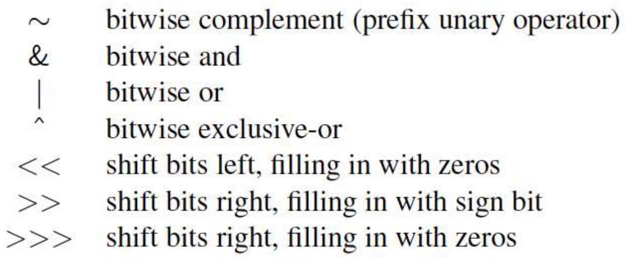
```
public class LeapYear {
public static void main(String[] args) {
      int year = Integer.parseInt(args[0]);
      boolean isLeapYear;
      // divisible by 4 but not 100
      isLeapYear = (year % 4 == 0) && (year % 100 != 0);
      // or divisible by 400
      isLeapYear = isLeapYear || (year % 400 == 0);
      System.out.println(isLeapYear);
} }
```

```
% java LeapYear 2004
 true
% java LeapYear 1900
 false
% java LeapYear 2000
 true
```

28

Bitwise Operators

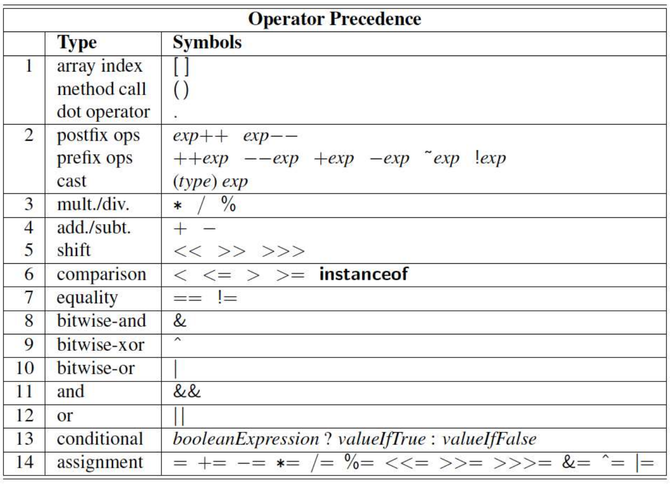
- Java provides the following bitwise operators for integers and booleans:



~	bitwise complement (prefix unary operator)
&	bitwise and
|	bitwise or
^	bitwise exclusive-or
<<	shift bits left, filling in with zeros
>>	shift bits right, filling in with sign bit
>>>	shift bits right, filling in with zeros

29

Operator Precedence



Operator Precedence

	Type	Symbols
1	array index method call dot operator	[ ] ( ) .
2	postfix ops prefix ops cast	*exp*++ *exp*−− ++*exp* −−*exp* +*exp* −*exp* ˜*exp* !*exp* (*type*) *exp*
3	mult./div.	∗ / %
4	add./subt.	+ −
5	shift	<< >> >>>
6	comparison	< <= > >= **instanceof**
7	equality	== !=
8	bitwise-and	&
9	bitwise-xor	ˆ
10	bitwise-or	|
11	and	&&
12	or	||
13	conditional	*booleanExpression* ? *valueIfTrue* : *valueIfFalse*
14	assignment	= += −= ∗= /= %= <<= >>= >>>= &= ˆ= |=

30