BIRZEIT UNIVERSITY

Elementary Programming

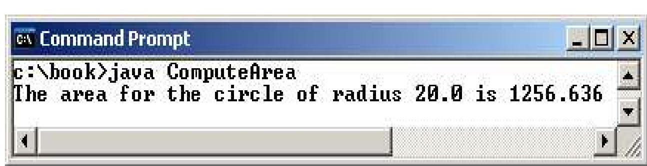
By: Mamoun Nawahdah (PhD)
2022

Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " + radius + " is " + area);
  }
}
```

memory

radius	20
area	1256.636



Identifiers

- Identifiers are for **naming** variables, methods, classes
- An **identifier** is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs (**$**).
- An identifier must start with a letter, an underscore (_), or a dollar sign (**$**). It cannot start with a digit.
- An identifier cannot be a reserved word.
- An identifier cannot be **true**, **false**, or **null**.
- An identifier can be of any length.

3

Java Keywords and Reserved Words

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

Variables

- ❖ Variables are used to represent values that may be changed in the program.
- ❖ A variable must be declared before it can be assigned a value.
- ❖ A variable declared in a method must be assigned a value before it can be used.

5

Declaring Variables

```
int x;             // Declare x to be an integer variable
double radius;     // Declare radius to be a double variable
char a;            // Declare a to be a character variable
```

Assignment Statements

```
x = 1;          // Assign 1 to x
radius = 1.0;   // Assign 1.0 to radius
a = 'A';        // Assign 'A' to a
```

6

Declaring and Initializing in 1 Step

```
int x = 1;
double d = 1.4;
```

Named Constants

```
final datatype CONSTANTNAME = VALUE;
final double PI = 3.14159;
final int SIZE = 3;
```


Naming Conventions

- Choose **meaningful** and descriptive names
- Variables and method names:
  - Use lowercase.
  - If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name.
  - For example, the variables **radius** and **area**, and the method **computeArea**.

Naming Conventions, cont.

❖ **Class names:**
- Capitalize the 1st letter of each word in the name
- For example, the class name **ComputeArea**

❖ **Constants**:
- Capitalize all letters in constants, and use underscores to connect words.
- For example, the constant **PI** and **MAX_VALUE**

Numerical Data Types

Name	Range	Storage Size
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
−	Subtraction	34.0 − 0.1	33.9
*	Multiplication	300*30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Integer Division

- 5 **/** 2 yields an integer 2.
- 5.0 **/** 2 yields a double value 2.5
- 5 **%** 2 yields 1 (the remainder of the division)
- The **%** operator is often used for positive integers, but it can also be used with negative integers and floating-point values.
- The remainder is negative only if the dividend is negative. For example,
  - -7 % 3 yields -1 -12 % 4 yields 0
  - -26 % -8 yields -2 20 % -13 yields 7

12

double vs. float

The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`

7 digits

13

Scientific Notation

- Floating-point literals can be written in scientific notation in the form of ***a*** *** 10^*b***. For example:
  - The scientific notation for 123.45 is 1.2345 * 10^2
  - For 0.012345 is 1.2345 * 10^-2
- A special syntax is used to write scientific notation numbers. For example:
  - 1.2345 * 10^2 is written as **1.2345E2** or **1.2345E+2**
  - 1.2345 * 10^-2 as **1.2345E-2**
- **E** (or **e**) represents an exponent, and can be in either lowercase or uppercase.

Evaluating Expressions

❖ Java expressions are evaluated in the same way as arithmetic expressions.

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

```
(3 + 4 * x) / 5 – 10 * (y - 5) * (a + b + c) /
        x + 9 * (4 / x + (9 + x) / y)
```

Operator Precedence

❖ Operators contained within pairs of parentheses **()** are evaluated first.

❖ When more than one operator is used in an expression, the following operator precedence rule is used to determine the order of evaluation:

- *****, **/**, and **%** operators are applied first.
- If an expression contains several *****, **/**, and **%** operators, they are applied from **left** to **right**.
- **+** and **-** operators are applied last.
- If an expression contains several **+** and **-** operators, they are applied from left to right.

Augmented Assignment Operators

Operator	Name	Example	Equivalent
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /=  8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

Note: There are no spaces in the augmented assignment operators.

17

Increment and Decrement Operators

Operator	Name	Description	Example (assume i = 1)
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`

18

Numeric Type Conversion

Consider the following statements:

byte i = 100;

long k = i * 3 + 4;

double d = i * 3.1 + k / 2;

Conversion Rules

- ❖ When performing a binary operation involving 2 operands of different types, Java automatically converts the operand using the following rules:

1. If one of the operands is **double**, the other is converted into double.
2. Otherwise, if one of the operands is **float**, the other is converted into float.
3. Otherwise, if one of the operands is **long**, the other is converted into long.
4. Otherwise, both operands are converted into **int**.

Type Casting

Implicit casting

double d = 3; (type widening)

Explicit casting

int i = (int) 3.0; (type narrowing)
int i = (int) 3.9; (Fraction part is truncated)

What is wrong? int x = 6 / 2.0;

range increases →

byte, short, int, long, float, double

21

Character Data Type

char letter = 'A'; (ASCII)

char numChar = '4'; (ASCII)

char letter = '**¥u**0041'; (Unicode)

char numChar = '**¥u**0034'; (Unicode)

NOTE: The increment and decrement operators can also be used on **char** variables to get the next or preceding Unicode character.

For example, the following statements display character **b**.

```
char ch = 'a';
System.out.println( ++ch );
```

22

ASCII Code for Commonly Used Characters

Characters	Code Value in Decimal	Unicode Value
'0' to '9'	48 to 57	\u0030 to \u0039
'A' to 'Z'	65 to 90	\u0041 to \u005A
'a' to 'z'	97 to 122	\u0061 to \u007A

Escape Sequences for Special Characters

Escape Sequence	*Name*	*Unicode Code*	*Decimal Value*
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Carriage Return	\u000D	13
\\\\	Backslash	\u005C	92
\\"	Double Quote	\u0022	34

Casting between char and Numeric Types

```
int i = 'a';      // Same as int i = (int)'a';

char c = 97;      // Same as char c = (char)97;
```

Comparing and Testing Characters

```
if (ch >= 'A' && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");
else if (ch >= 'a' && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");
else if (ch >= '0' && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

The String Type

❖ The char type only represents **1** character.

❖ To represent a string of characters, use the data type called **String**. For example:

```
String message = "Welcome to Java!";
```

❖ String is actually a predefined class in the Java library.

❖ The String type is not a primitive type. It is known as a ***reference*** *type*.

String Concatenation

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

Simple Methods for Strings

Method	Description
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase.
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

Console Input

- You can use the **Scanner** class for console input
- Java uses **System.in** to refer to the standard input device (i.e. Keyboard)

```
import  java.util.Scanner;
public class Test{
   public static void main(String[] s){
      Scanner input = new Scanner(System.in);
      System.out.println("Enter  X : ");
       int x = input.nextInt();
       System.out.println("You entered: "+ x);
   }
}
```

30

Reading Numbers from the Keyboard

Method	Description
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

31

Reading a String from the Console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```