Universidad
Carlos III de Madrid
www.uc3m.es

Lesson 2

Data and Operators

Programming

Grade in Computer Science

Outline

1. **Basic data types and variables**
2. **Input and output**
3. **Comments**
4. **Arrays**
5. **Operations with data**
6. **Casting between data types**
7. **Enumerates**
8. **Classes as data structures**

Outline

1. **Basic data types and variables**
2. Arrays
3. Input and output
4. Comments
5. Operations with data
6. Casting between data types
7. Enumerates
8. Classes as data structures

Data

Information processed by the program

read from the keyboard

used in calculations

printed on the screen

written on a file

…

Literals

Values directly introduced in the program

Variables

Symbols whose value change during the program execution

> piece of memory with a readable name

Literals

Integers

int
long
short
byte

Real

float
double

Characters

char

Boolean

boolean

String

String

Type	Example
int	-2147483648, 2147483647
long	-85738593L, 8593854L
short	-30000, 8438, -4923
byte	-32, 123, 39
float	-3.56E+30F, 8.234
double	-2.49E+300, 3.95E+200
char	‘a’, ‘D’, ‘\n’, ‘\\’, ‘\”’
boolean	true, false
String	"hello world!"

Integers

Signed (positive and negative integer values)

Four types: **`byte`**, **`short`**, **`int`**, **`long`**

Range is platform independent

By default integers are of type `int`

For a long append an `L`

Example

- `int`

```
123456
-156
```

- `long`

```
123456L
989493849859L
-284829848L
```

Real (floating point)

Two types: **`float`**, **`double`**

By default floats are of type `double`

For a float append an `F`

Example

- > `double`
  ```
  123.45
  -18.23
  3.14E-5
  ```
- > `float`
  ```
  123.45F
  3.45E+21F
  -284829848F
  ```

Special values for **`float`** and **`double`**:

Infinity (`Inf`), -Infinity (`-Inf`), not a number (`NaN`)

These values may appear as a result of an operation, but cannot be directly assigned

Characters

Enclosed between single quotes: `'a'`, `'A'`

Escape characters: `'\''`, `'\b'`, `'\t'`, `'\n'`, `'\\'`, …

UNICODE 16 bits

Each characters has an equivalent numerical code, defined by the UNICODE standard

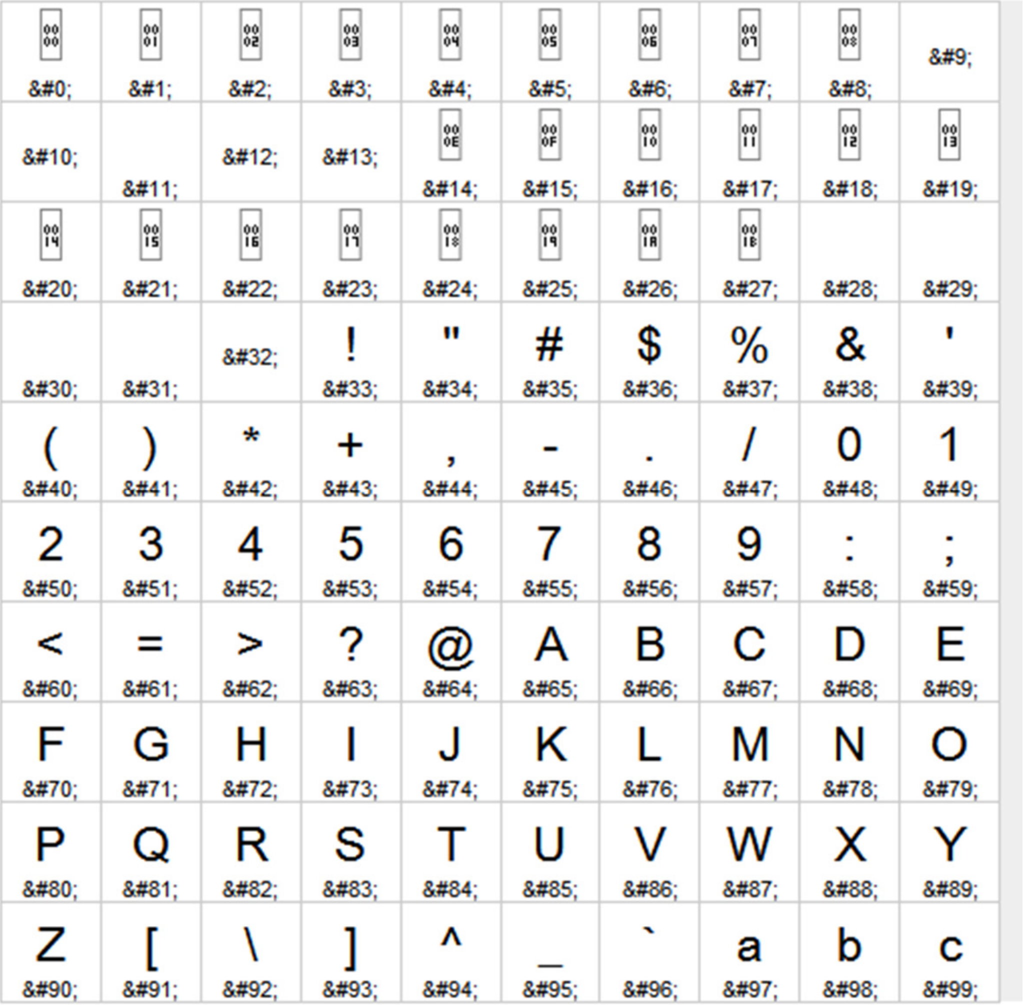
Unicode code `'\u0065'` corresponds to `'A'`

Characters and integers can be interchanged in some cases

Integer value 65 corresponds to `'A'`

1. Basic data types and variables

UNICODE table



0000 &#0;	0001 &#1;	0002 &#2;	0003 &#3;	0004 &#4;	0005 &#5;	0006 &#6;	0007 &#7;	0008 &#8;	&#9;
&#10;	&#11;	&#12;	&#13;	000E &#14;	000F &#15;	0010 &#16;	0011 &#17;	0012 &#18;	0013 &#19;
0014 &#20;	0015 &#21;	0016 &#22;	0017 &#23;	0018 &#24;	0019 &#25;	001A &#26;	001B &#27;	&#28;	&#29;
&#30;	&#31;	&#32;	! &#33;	" &#34;	# &#35;	$ &#36;	% &#37;	& &#38;	' &#39;
(&#40;	) &#41;	* &#42;	+ &#43;	, &#44;	- &#45;	. &#46;	/ &#47;	0 &#48;	1 &#49;
2 &#50;	3 &#51;	4 &#52;	5 &#53;	6 &#54;	7 &#55;	8 &#56;	9 &#57;	: &#58;	; &#59;
< &#60;	= &#61;	> &#62;	? &#63;	@ &#64;	A &#65;	B &#66;	C &#67;	D &#68;	E &#69;
F &#70;	G &#71;	H &#72;	I &#73;	J &#74;	K &#75;	L &#76;	M &#77;	N &#78;	O &#79;
P &#80;	Q &#81;	R &#82;	S &#83;	T &#84;	U &#85;	V &#86;	W &#87;	X &#88;	Y &#89;
Z &#90;	[&#91;	\ &#92;	] &#93;	^ &#94;	_ &#95;	` &#96;	a &#97;	b &#98;	c &#99;

Source: http://www.ftrain.com/unicode/

Strings

Strings are complex data types to represent and manage a string of characters

Enclosed between double quotes " " (shift + 2 key)

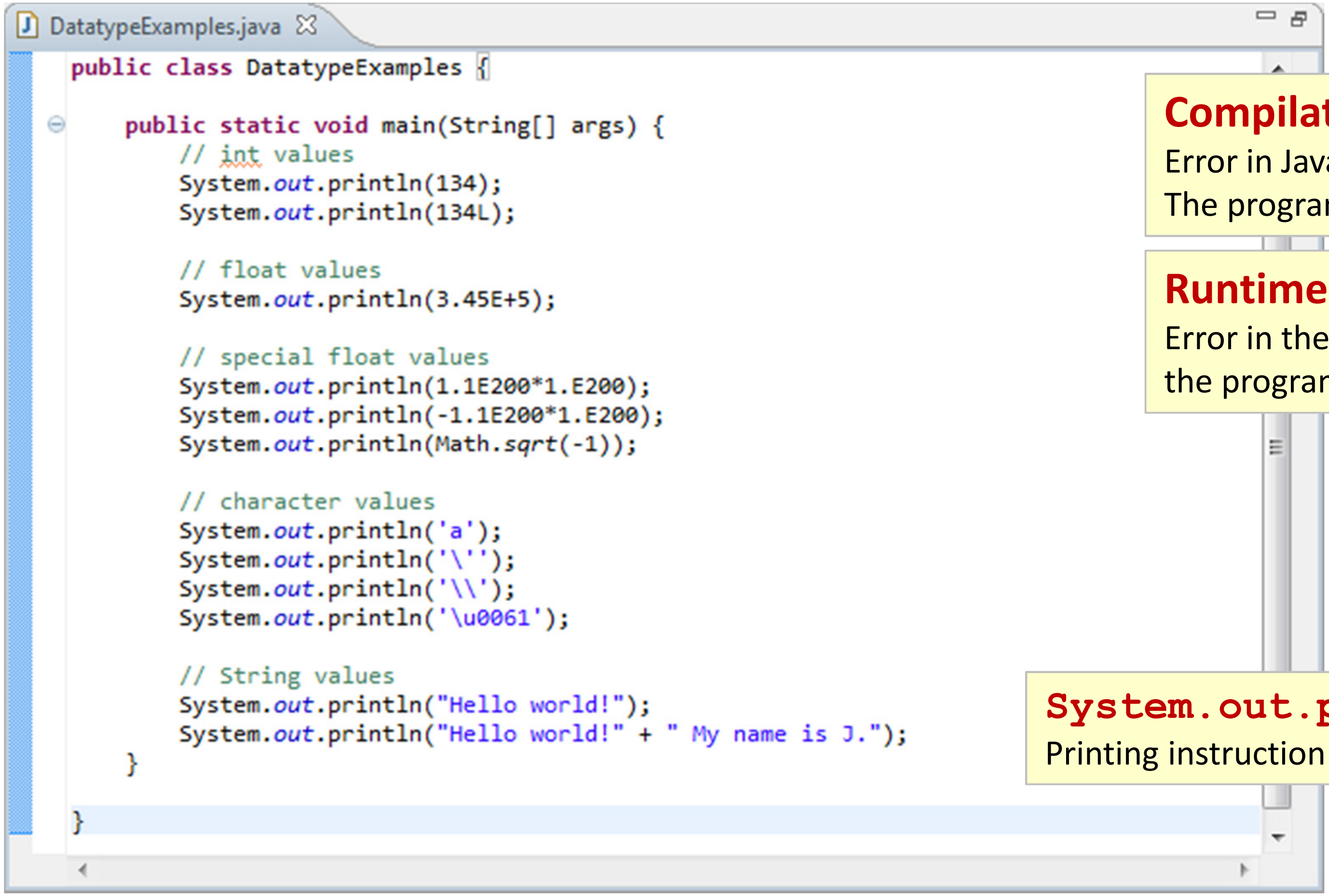
```
"Hello world!"

"My name is Bond"
```

Strings can be concatenated with the **+** operator

```
"My name is Bond"
```

1. Basic data types and variables

Literals example



DatatypeExamples.java

```
public class DatatypeExamples {

    public static void main(String[] args) {
        // int values
        System.out.println(134);
        System.out.println(134L);

        // float values
        System.out.println(3.45E+5);

        // special float values
        System.out.println(1.1E200*1.E200);
        System.out.println(-1.1E200*1.E200);
        System.out.println(Math.sqrt(-1));

        // character values
        System.out.println('a');
        System.out.println('\'');
        System.out.println('\\');
        System.out.println('\u0061');

        // String values
        System.out.println("Hello world!");
        System.out.println("Hello world!" + " My name is J.");
    }

}
```

Compilation error
Error in Java syntax
The program cannot run

Runtime error
Error in the execution of the program

`System.out.println`
Printing instruction

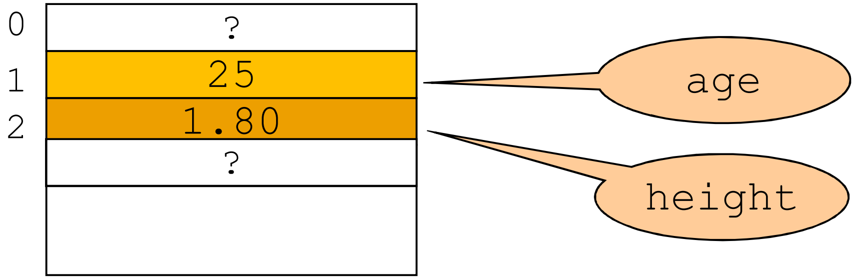
Comments
Notes to the code

Variables store data that can be changed during the execution of a program

Can be seen as a piece of the memory to store a piece of data

User-defined readable name for a cell of the memory

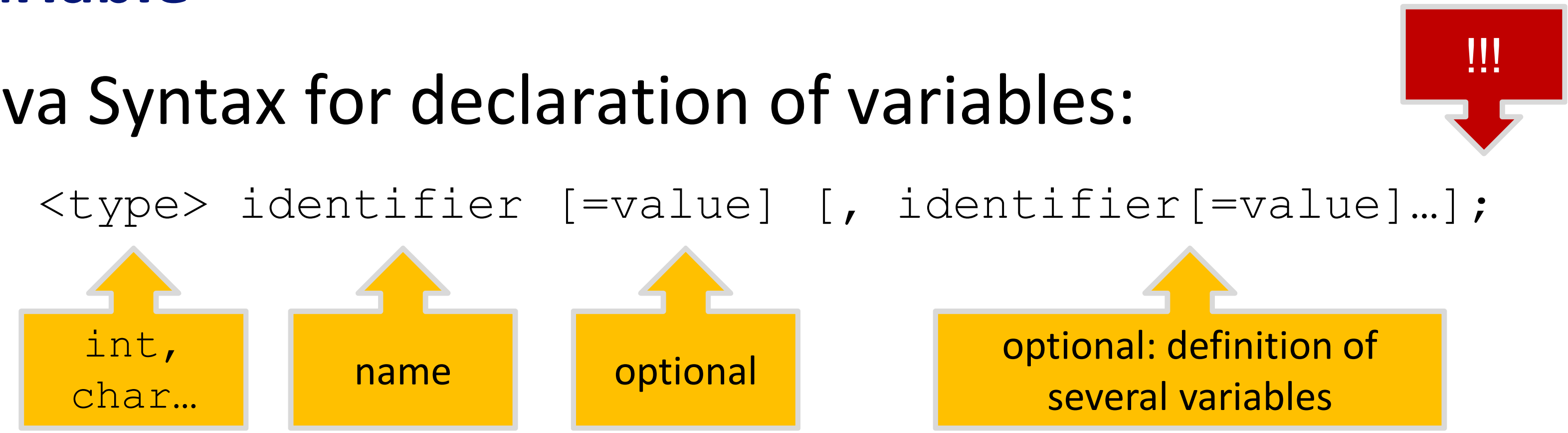
When the name (or identifier) of the variable is used in the program, the information at the address of the variable is accessed



Variables store data that can be changed during the execution of a program

Java is a *strongly typed language*: Necessary to **declare a variable before it is used and define the type of the variable**

Java Syntax for declaration of variables:



!!!

```
<type> identifier [=value] [, identifier[=value]…];
```

`int, char…`

name

optional

optional: definition of several variables

[ ] optional
<> compulsory

1. Basic data types and variables

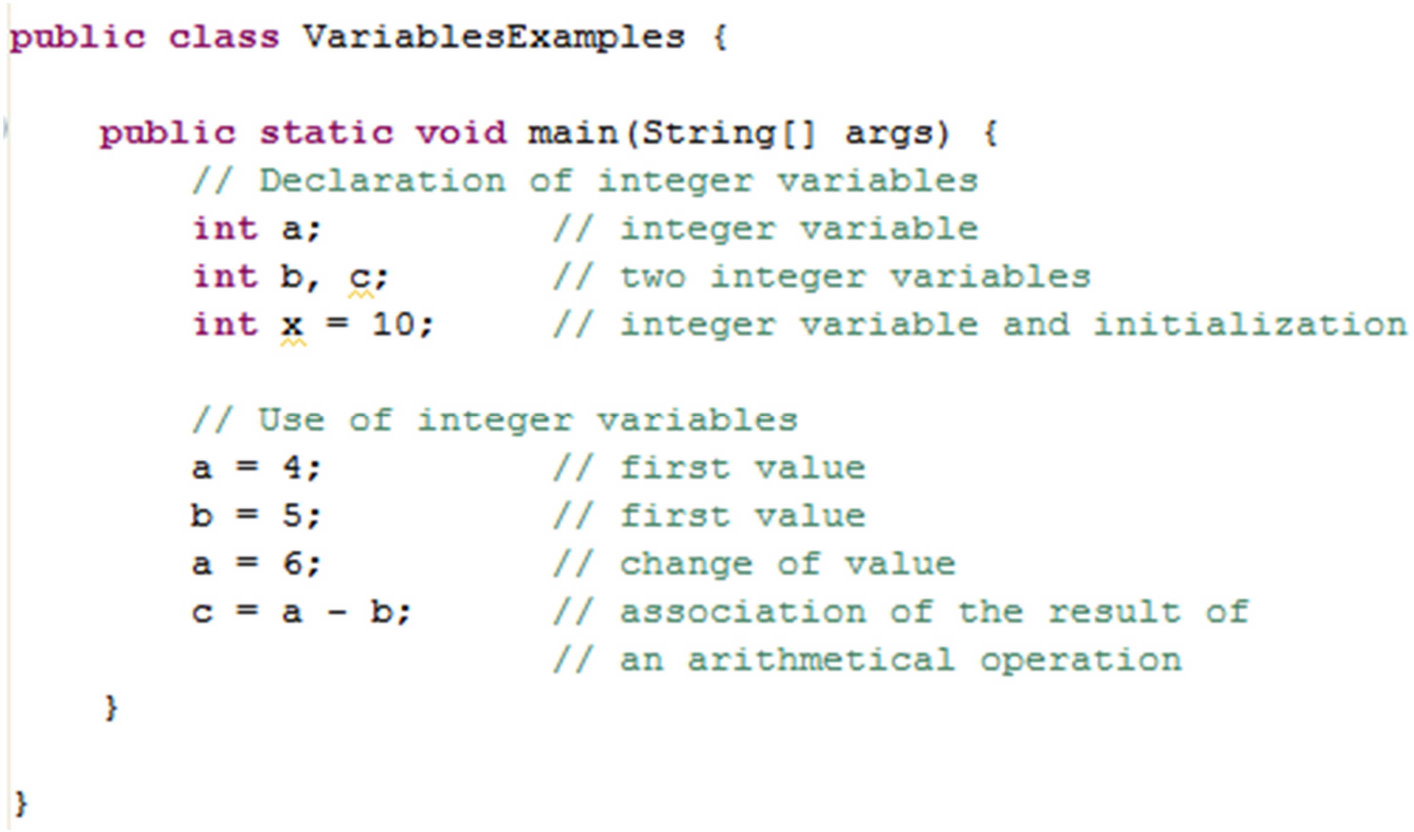
Variable types

Type	Contains	Default	Size	Range
`boolean`	`true` or `false`	`false`	1 bit	NA
`char`	Unicode character	'\u0000'	16 bits	'\u0000' to '\uFFFF'
`byte`	Signed integer	0	8 bits	-128 to 127
`short`	Signed integer	0	16 bits	-32768 to 32767
`int`	Signed integer	0	32 bits	-2147483648 to 2147483647
`long`	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
`float`	IEEE 754 floating point	0.0	32 bits	±1.4E-45 to ±3.4028235E+38
`double`	IEEE 754 floating point	0.0	64 bits	±4.9E-324 to ±1.7976931348623157E+308
`String`	Unicode character string	Empty string	-	-

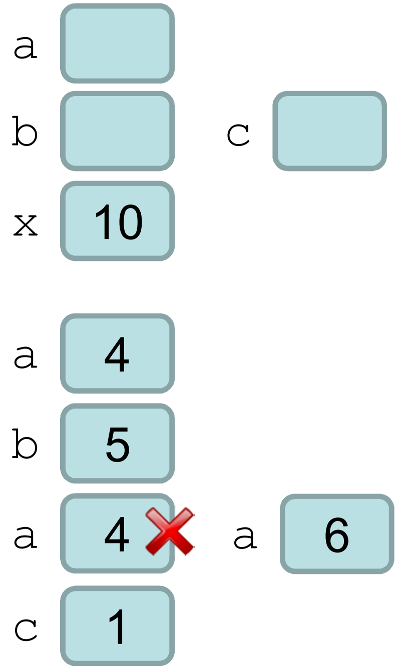
1. Basic data types and variables

Variable declaration and assignation

VariablesExamples.java



```
public class VariablesExamples {

    public static void main(String[] args) {
        // Declaration of integer variables
        int a;              // integer variable
        int b, c;           // two integer variables
        int x = 10;         // integer variable and initialization

        // Use of integer variables
        a = 4;              // first value
        b = 5;              // first value
        a = 6;              // change of value
        c = a - b;          // association of the result of
                            // an arithmetical operation
    }

}
```



Variable declaration
Memory is allocated

Variable initialization
First value assignment

Variable definition
Declaration + initialization

1. Basic data types and variables

Variable scope

Variables are not valid in a whole *program*

- Names can be reused
- Side-effects are avoided

Scope: Section of the code where the variable is valid and can be used

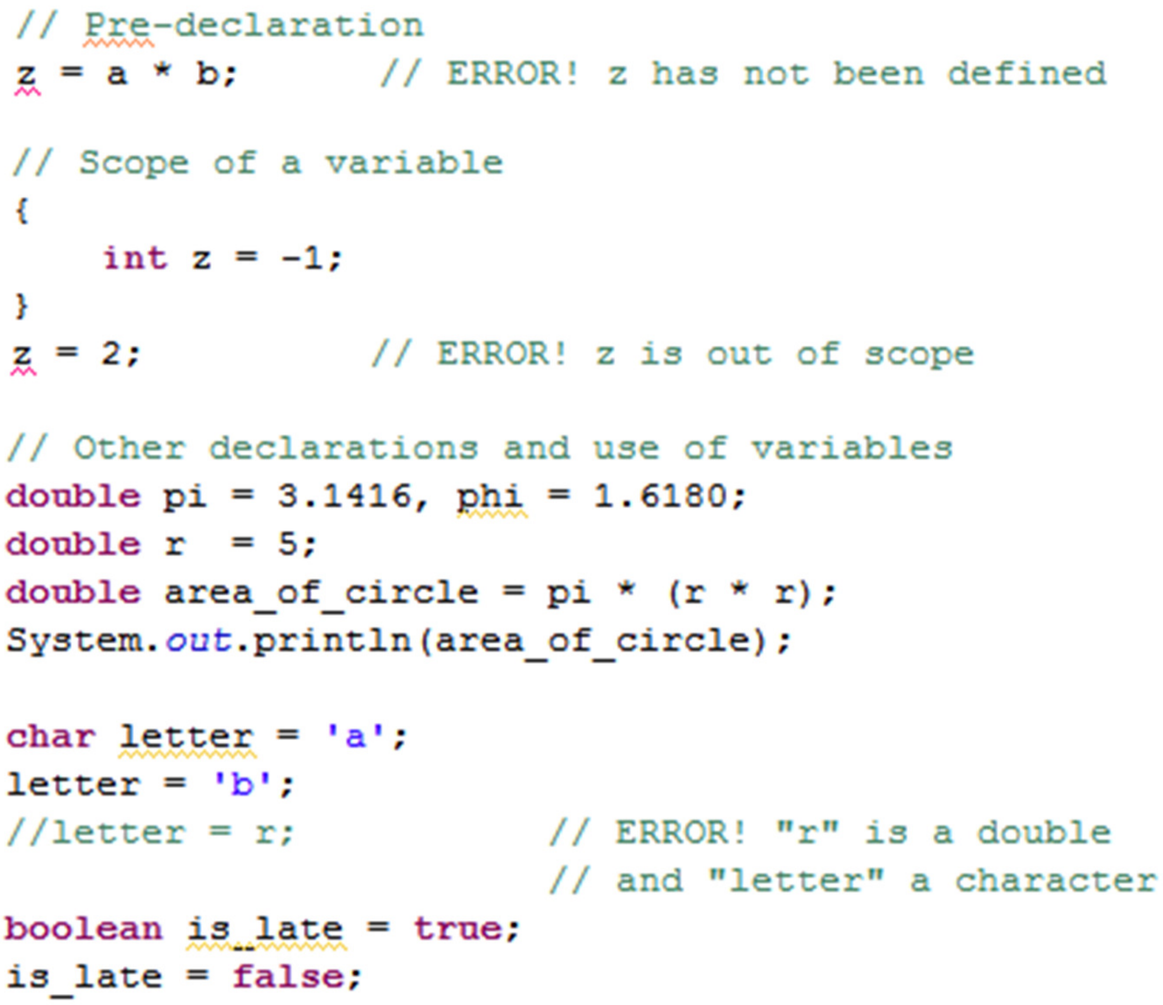
The scope of a variable encompasses is the block of code in which it is declared

- A block is delimited by braces { }
  - Also named curly brackets

1. Basic data types and variables

Variable declaration and assignments

VariablesExamples.java



```
// Pre-declaration
z = a * b;       // ERROR! z has not been defined

// Scope of a variable
{
    int z = -1;
}
z = 2;           // ERROR! z is out of scope

// Other declarations and use of variables
double pi = 3.1416, phi = 1.6180;
double r  = 5;
double area_of_circle = pi * (r * r);
System.out.println(area_of_circle);

char letter = 'a';
letter = 'b';
//letter = r;              // ERROR! "r" is a double
                           // and "letter" a character
boolean is_late = true;
is_late = false;
```

Variable assignment

Variables can be assigned to values with different types only under certain conditions

1. Basic data types and variables

Constants

Special variables whose value cannot be changed during the execution of the program

Use **final** in the declaration of a variable to make it constant:

```
final <type> <identifier> [= value];
```

Constants are used as variables

E.g.:

```
final double PI = 3.14;
double r = 5;
double a = 2 * PI * r;
```

The value of a constant **can be modified only once**! Otherwise, we get a compilation error.

Outline

1. Basic data types and variables
2. **Arrays**
3. Input and output
4. Comments
5. Operations with data
6. Casting between data types
7. Enumerates
8. Classes as data structures

Arrays are collections of elements of the same type which are collectively *managed*

Creation

Syntax for declaration of one-dimensional arrays

```
<type> [] <identifier>;
```

E.g.:

```
int [] myArray;
```

Syntax for initialization of one-dimensional arrays

```
<identifier> = new <type>[<n° of elements>];
```

E.g.:

```
myArray = new int[10];
```

Array elements have a default value

Array elements do not have to be initialized before using them in an expression

Default values are 0 for numbers and characters, false for booleans, null for Strings

Syntax for accessing values

```
<identifier>[<position>];
```

E.g.:

```
System.out.println(myArray[2]);
```

Syntax for value assignment

```
identifier[<position>] = <value>;
```

E.g.:

```
myArray[3] = 28;
```

Syntax for multi-value assignment (only in initialization)

```
identifier = new <type>[] {<list of values>};
```

E.g.:

```
myArray = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Use `length` to get the **size of an array**

E.g.:

```
System.out.println(myArray.length)
```

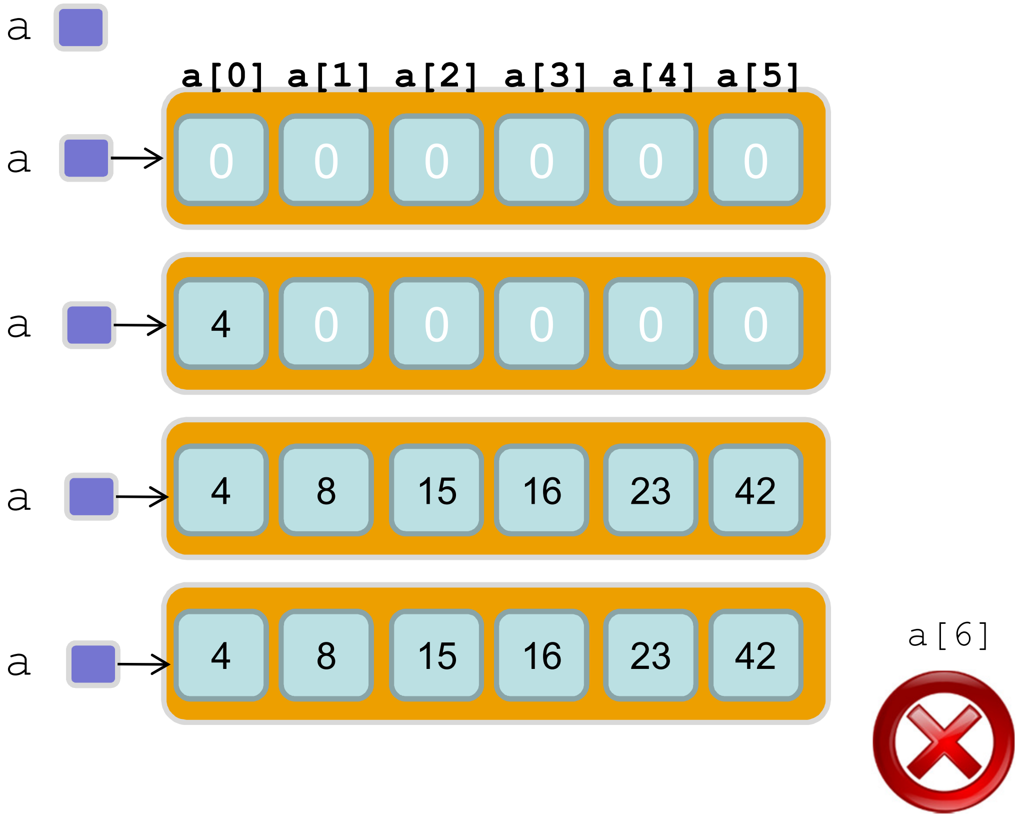
2. Arrays

Basic array example

```
int [] a;
a = new int[6];
a[0] = 4;
a[1] = 8;
a[2] = 15;
a[3] = 16;
a[4] = 23;
a[5] = 42;

a[6] = 100; ✗
```

Array index out of bounds

Accessing to a non-allocated position of an array is a serious mistake resulting in a runtime error



Runtime error!

ArraysExamples.java

Syntax for array assignment

Contents are not copied in a direct assignment! Both identifiers refers to the same array

```
<identifier1> = <identifier2>;
```

E.g.:

```
myArray_1 = myArray_2;
```

Syntax for array copy

Contents are copied! Both identifiers refers to different arrays

Option 1: `<identifier1>[index] = <identifier2>[index];`

E.g.:

```
myArray_1[0] = myArray_2[0];
```

Option 2: `System.arrayCopy(`*`source_array`*`, `*`source_position`*`, `*`destination_array`*`, `*`destination_position`*`, `*`n_elements_to_copy`*`)`

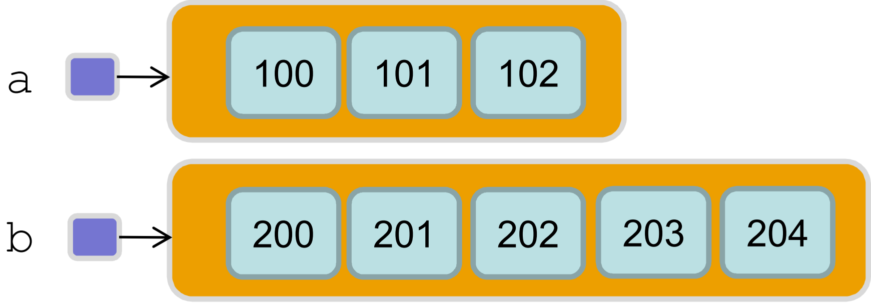
E.g.:

```
System.arrayCopy(myArray_2, 0, myArray_1, 0, myArray_1.length);
```

2. Arrays

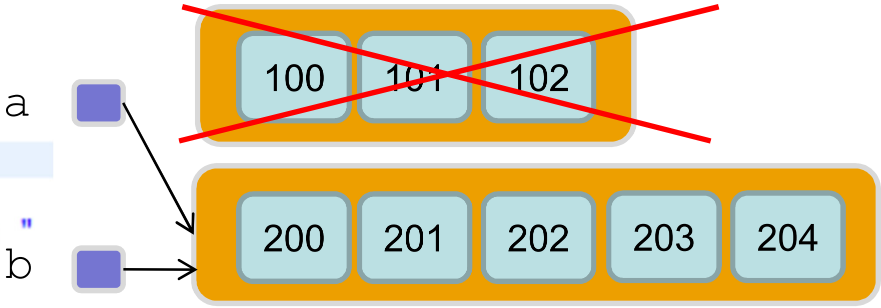
ArraysExamples.java

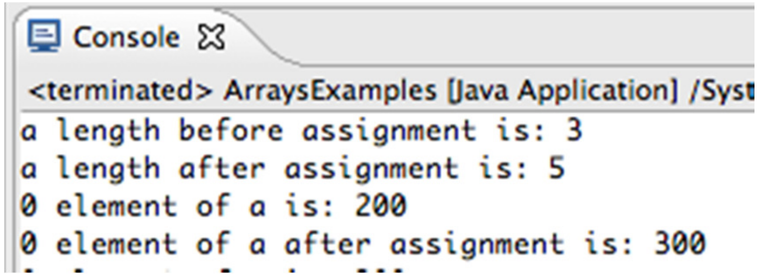
```
// Change size and values
a = new int[] {100, 101, 102};
System.out.println("0 new element: " + a[0]);
System.out.println("1 new element: " + a[1]);
System.out.println("2 new element: " + a[2]);
```



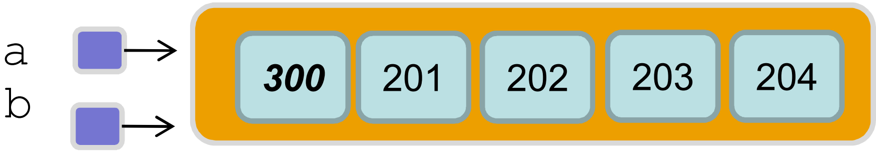
```
 // Array assignment
 int [] b = new int[5];
 b[0] = 200;
 b[1] = 201;
 b[2] = 202;
 b[3] = 203;
 b[4] = 204;
```

```
System.out.println("a length before assignment is: "
        + a.length);
a = b;
System.out.println("a length after assignment is: "
        + a.length);
System.out.println("0 element of a is: "
        + a[0]);
b[0] = 300;
System.out.println("0 element of a after assignment is: "
        + a[0]);
```



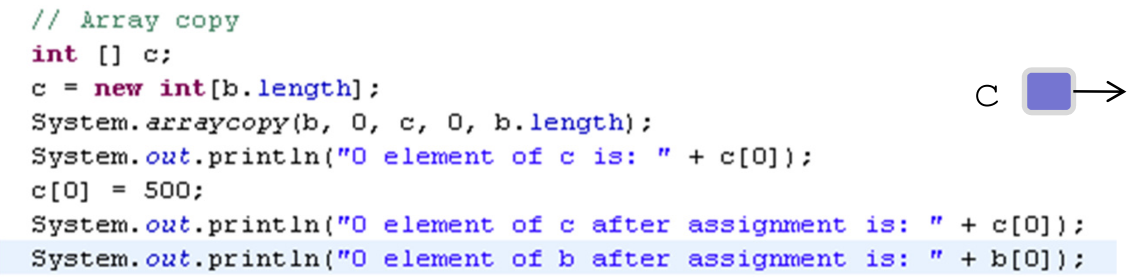


```
<terminated> ArraysExamples [Java Application] /Syst
a length before assignment is: 3
a length after assignment is: 5
0 element of a is: 200
0 element of a after assignment is: 300
```

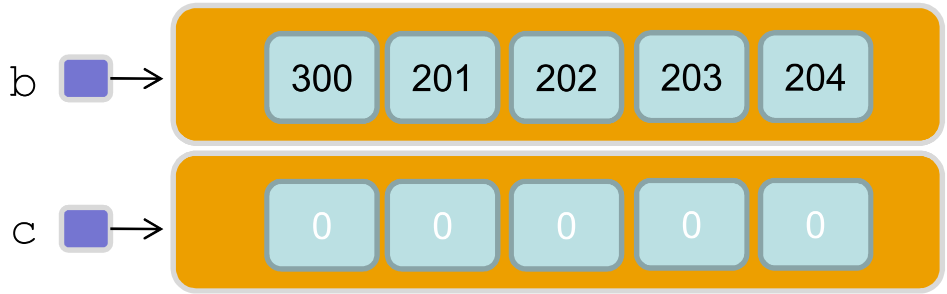


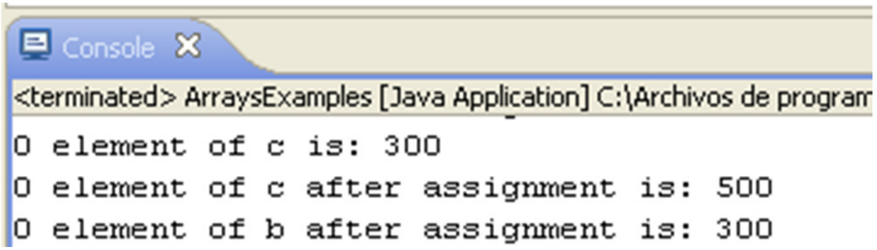
2. Arrays

Array copy example

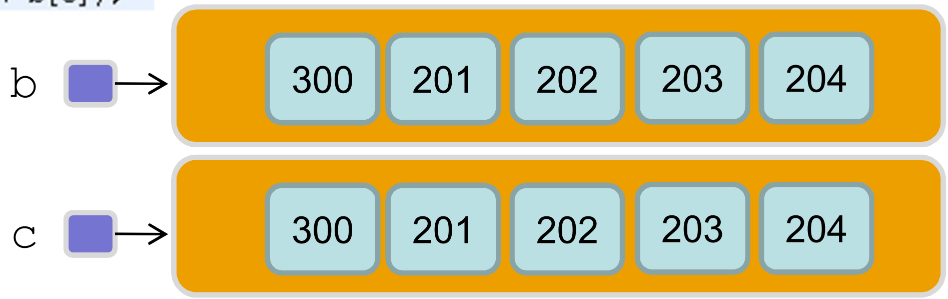


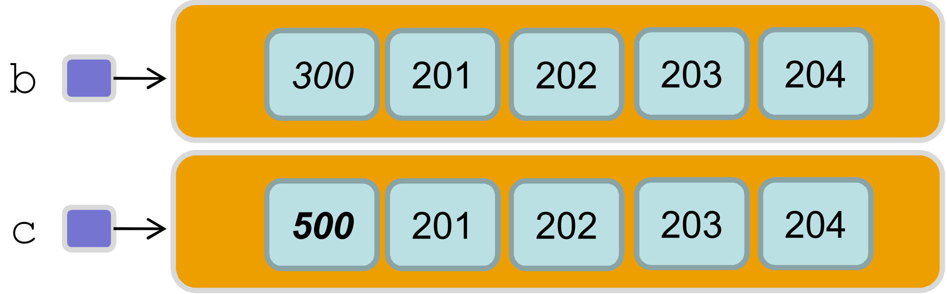
```
// Array copy
int [] c;
c = new int[b.length];
System.arraycopy(b, 0, c, 0, b.length);
System.out.println("0 element of c is: " + c[0]);
c[0] = 500;
System.out.println("0 element of c after assignment is: " + c[0]);
System.out.println("0 element of b after assignment is: " + b[0]);
```





```
<terminated> ArraysExamples [Java Application] C:\Archivos de program
0 element of c is: 300
0 element of c after assignment is: 500
0 element of b after assignment is: 300
```





ArraysExamples.java

Multi-dimension arrays can be also created

Syntax for declaration of two-dimensional arrays

```
<type> [][] identifier;
```

E.g.:

```
int [][] my2DMatrix;
```

Syntax for initialization of two-dimensional arrays

```
identifier = new <type>[<n° elements>][<n° elements>];
```

E.g.:

```
my2DMatrix = new int[3][3];
```

Syntax for value assignment of two-dimensional arrays

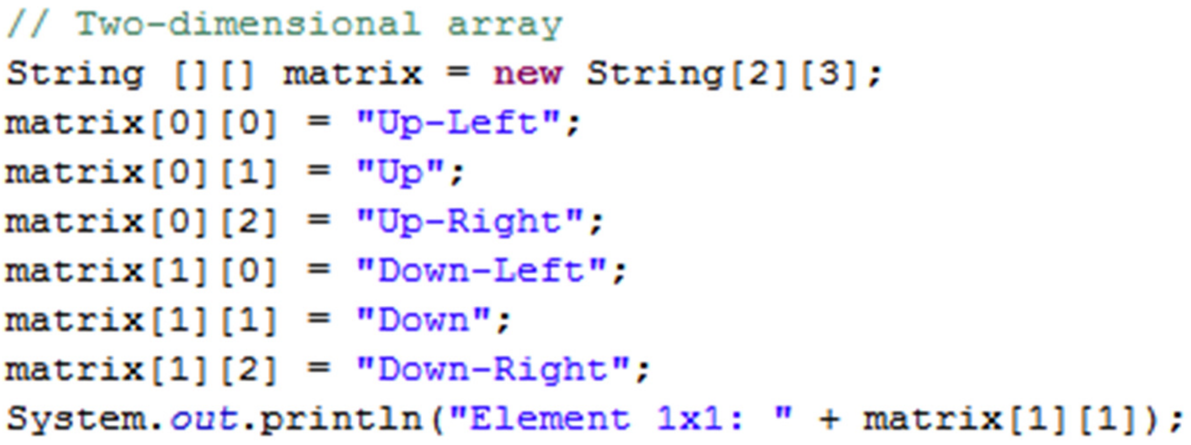
```
identifier[<position>][<position>] = <value>;
```

E.g.:

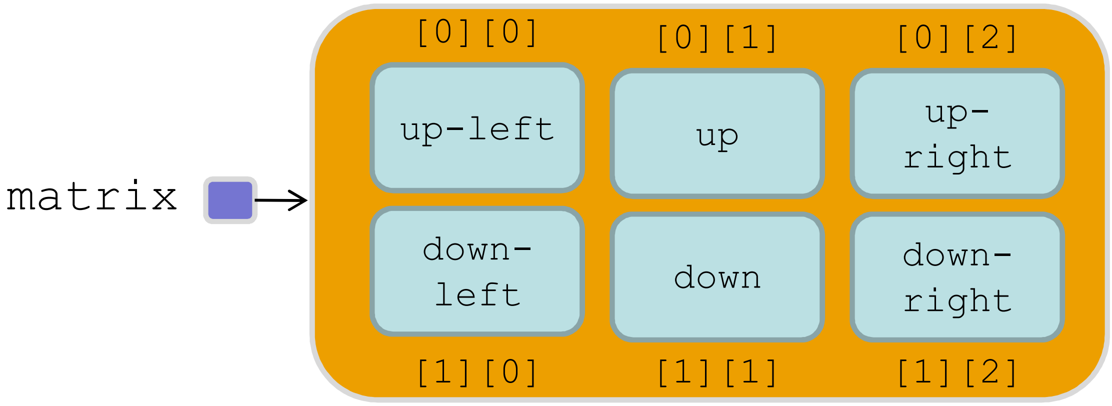
```
my2DMatrix[1][2] = 17;
```

Syntax and use can be **extended to n-dimension** arrays

Multi-dimension arrays example



```
// Two-dimensional array
String [][] matrix = new String[2][3];
matrix[0][0] = "Up-Left";
matrix[0][1] = "Up";
matrix[0][2] = "Up-Right";
matrix[1][0] = "Down-Left";
matrix[1][1] = "Down";
matrix[1][2] = "Down-Right";
System.out.println("Element 1x1: " + matrix[1][1]);
```



matrix

[0][0]	[0][1]	[0][2]
up-left	up	up-right
down-left	down	down-right
[1][0]	[1][1]	[1][2]

ArraysExamples.java

2. Arrays

Irregular arrays

Irregular arrays are arrays that have a different number of elements in each row

E.g.: A 2-dimensional array to store the names of the 1st year students, classified by group

Syntax for declaration of irregular two-dimensional arrays is the same as for regular arrays

```
<type> [][] <identifier>;
```

E.g.:

```
String [][] students;
```

Syntax for **initialization of irregular two-dimensional arrays is different!** Each row is created with a different new instruction.

```
<identifier> = new <type>[<n° rows>] [];
```

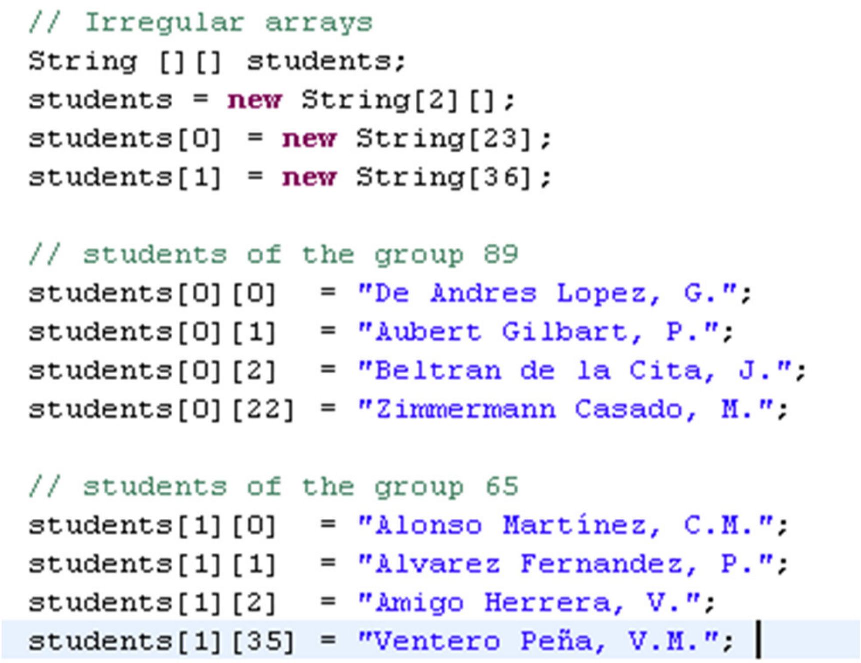
E.g.:

```
students    = new String[2][];
students[0] = new String[23];         // Students of grade in Computer Eng.
students[1] = new String[36];         // Students of grade in Comm. Syst.
```

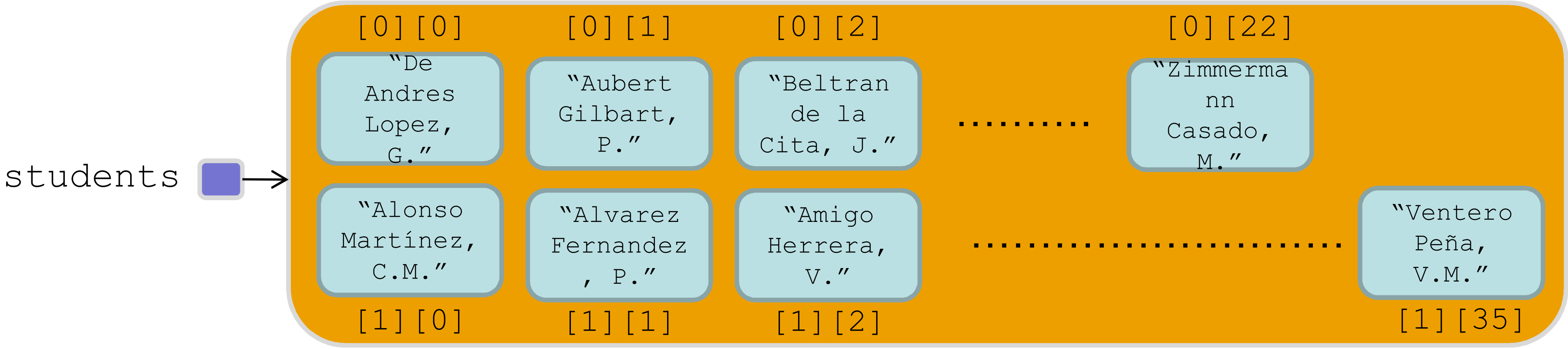
Syntax for accessing values is the same as for regular arrays, but we must be **careful with the size of the arrays**

3.1.5. Arrays

ArraysExamples.java



```
// Irregular arrays
String [][] students;
students = new String[2][];
students[0] = new String[23];
students[1] = new String[36];

// students of the group 89
students[0][0]  = "De Andres Lopez, G.";
students[0][1]  = "Aubert Gilbart, P.";
students[0][2]  = "Beltran de la Cita, J.";
students[0][22] = "Zimmermann Casado, M.";

// students of the group 65
students[1][0]  = "Alonso Martínez, C.M.";
students[1][1]  = "Alvarez Fernandez, P.";
students[1][2]  = "Amigo Herrera, V.";
students[1][35] = "Ventero Peña, V.M.";
```



Outline

1. Basic data types and variables
2. Arrays
3. **Input and output**
4. Comments
5. Operations with data
6. Casting between data types
7. Enumerates
8. Classes as data structures

System.out for printing on the screen

Methods for writing on screen:

Without a line jump

```
System.out.print(<String>);
```

With a line jump

```
System.out.println(<String>);
```

A line jump can be achieved by writing a new line character '\n'

> `println("Hi!")` is equivalent to `print("Hi!\n")`

Strings can be concatenated with the **+** operator within printing instructions

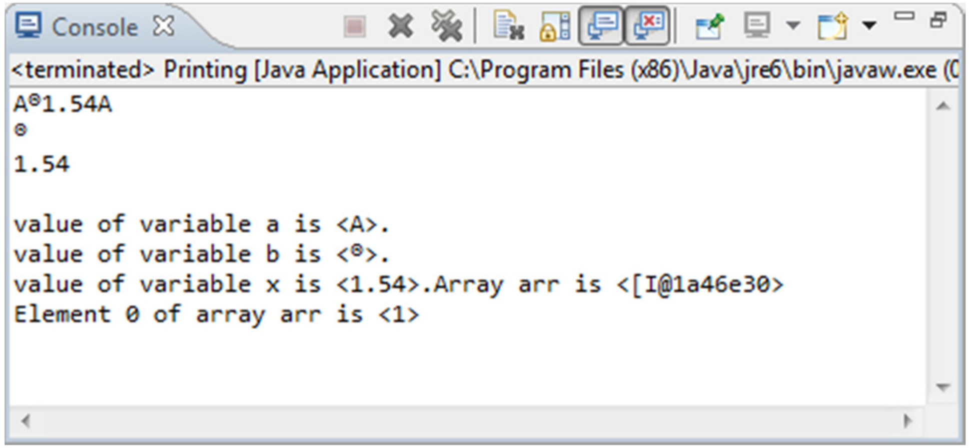
Other values with different datatypes can be appended with the + operator

> Java automatically converts them into the corresponding string

Arrays must be printed element-by-element!

3. Input and output

Printing on the screen



```
<terminated> Printing [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (C
A®1.54A
®
1.54

value of variable a is <A>.
value of variable b is <®>.
value of variable x is <1.54>.Array arr is <[I@1a46e30>
Element 0 of array arr is <1>
```

```java
char a = 'A';
char b = '\u00AE';
double x = 1.54;

// Without new line
System.out.print(a);
System.out.print(b);
System.out.print(x);

// With new line (after the printing)
System.out.println(a);
System.out.println(b);
System.out.println(x);

// Concatenation of strings and values
System.out.print("\nvalue of variable a is <" + a + ">.\n");
System.out.println("value of variable b is <" + b + ">.");
System.out.print("value of variable x is <" + x + ">.");

// Arrays
int [] arr = new int[] {1, 2, 3, 4};
System.out.println("Array arr is <" + arr + ">");
System.out.println("Element 0 of array arr is <" + arr[0] + ">");
```

Printing.java

`Scanner` class can be used to read values from the keyboard

Use:

1. Import `java.util.* package`

   ```
   import java.util.*;
   ```

2. Declare and initialize a `Scanner` object `sc`

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

3. Read values

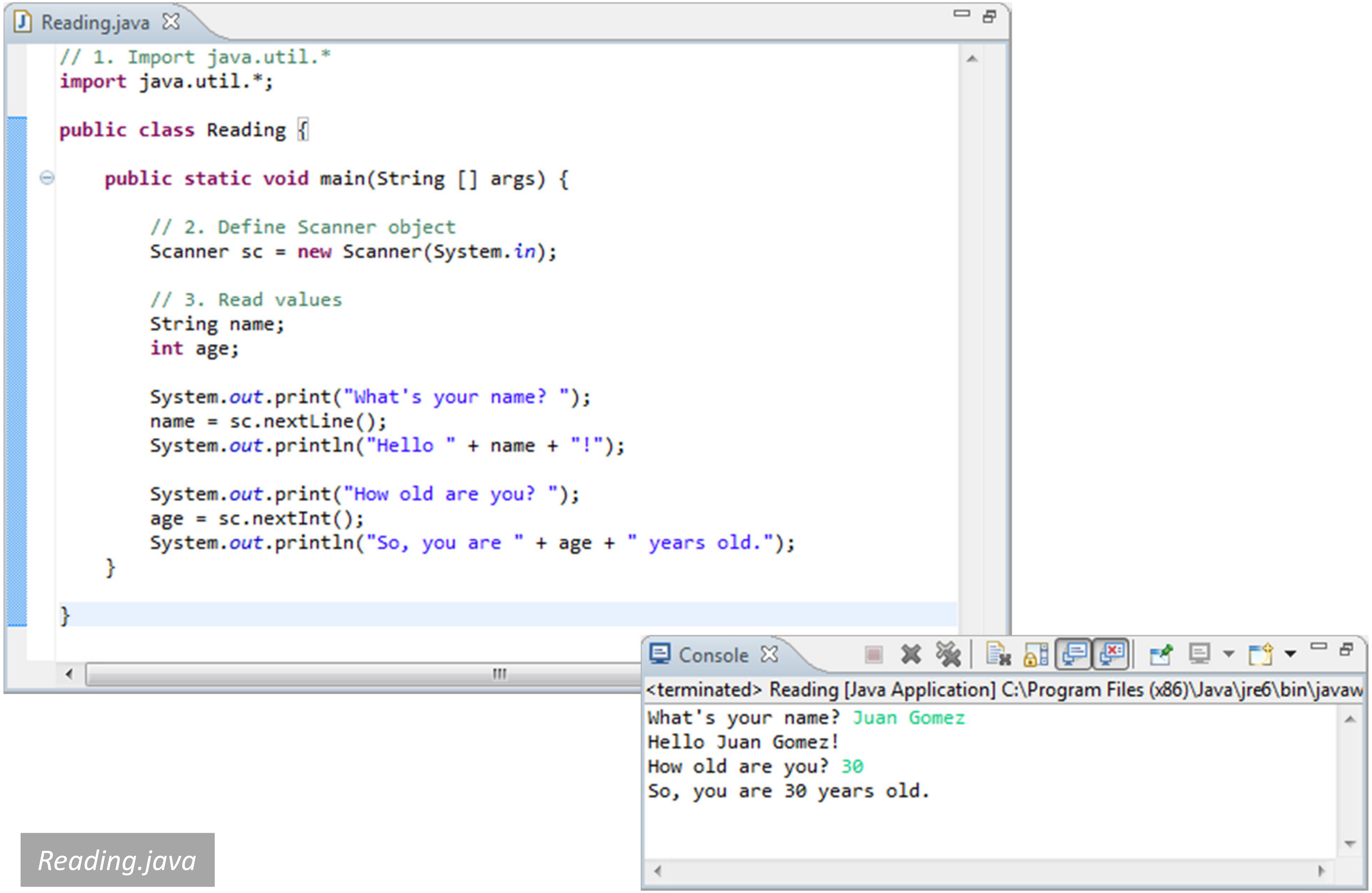
Integer:	`int a = sc.nextInt ();`	
Float:	`float b = sc.nextFloat();`	
Double:	`double c = sc.nextDouble();`	
String:	`String s = sc.next();`	(No blank spaces)
	`String s = sc.nextLine();`	(With blank spaces)

…

Universidad
Carlos III de Madrid
www.uc3m.es

3. Input and output

Reading from the keyboard



Reading.java

```java
// 1. Import java.util.*
import java.util.*;

public class Reading {

    public static void main(String [] args) {

        // 2. Define Scanner object
        Scanner sc = new Scanner(System.in);

        // 3. Read values
        String name;
        int age;

        System.out.print("What's your name? ");
        name = sc.nextLine();
        System.out.println("Hello " + name + "!");

        System.out.print("How old are you? ");
        age = sc.nextInt();
        System.out.println("So, you are " + age + " years old.");
    }

}
```

Console

```
<terminated> Reading [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw
What's your name? Juan Gomez
Hello Juan Gomez!
How old are you? 30
So, you are 30 years old.
```

Reading.java

Outline

1. Basic data types and variables
2. Arrays
3. Input and output
4. **Comments**
5. Operations with data
6. Casting between data types
7. Enumerates
8. Classes as data structures

4. Comments

Definition

Comments are notes to the code that are not executed

Its **very important to comment the code well**:

- Makes the code readable and understandable
- Although we now know perfectly what it does, perhaps within years we will have to reuse it
- Perhaps other programmers reuse our code and need to understand it
- It is a good practice to introduce a comment at the beginning of each file describing what it does

4. Comments

Types

Single line comments

Using the characters //

Everything appearing **on the right** is a comment, and it is ignored by the compiler

Multiple line comments

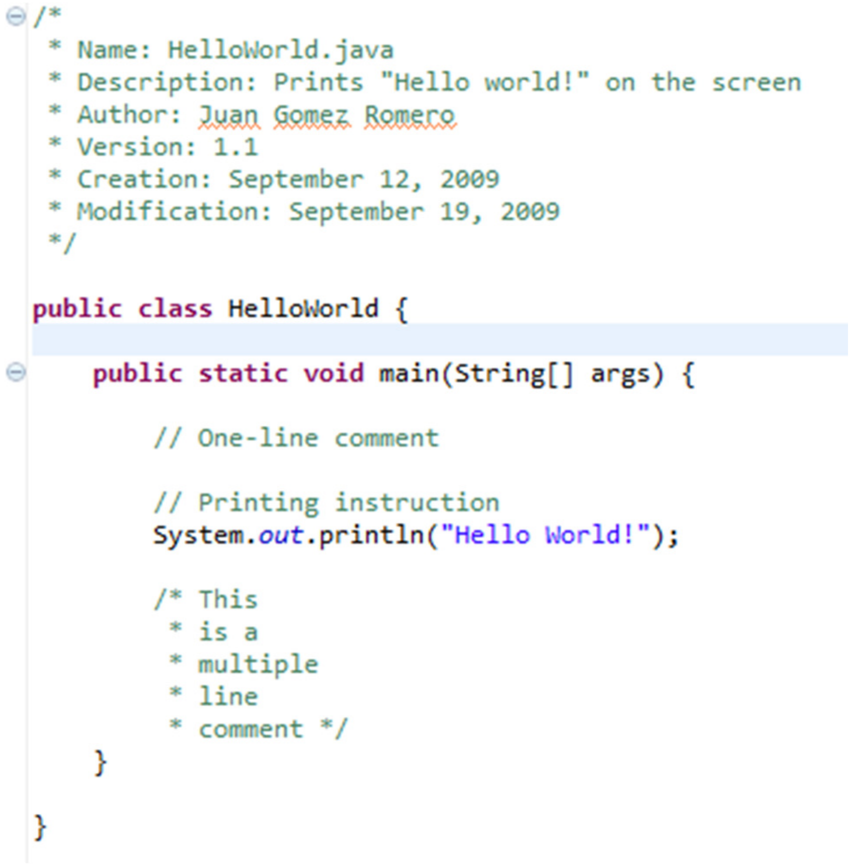
Using the characters /* for the beginning of the comment, and */ for the end

Everything written **in between** is a comment, and it is ignored by the compiler

4. Comments

Example

HelloWorld.java



```
/*
 * Name: HelloWorld.java
 * Description: Prints "Hello world!" on the screen
 * Author: Juan Gomez Romero
 * Version: 1.1
 * Creation: September 12, 2009
 * Modification: September 19, 2009
 */

public class HelloWorld {

    public static void main(String[] args) {

        // One-line comment

        // Printing instruction
        System.out.println("Hello World!");

        /* This
         * is a
         * multiple
         * line
         * comment */
    }

}
```

Outline

1. Basic data types and variables
2. Arrays
3. Input and output
4. Comments
5. **Operations with data**
6. Casting between data types
7. Enumerates
8. Classes as data structures

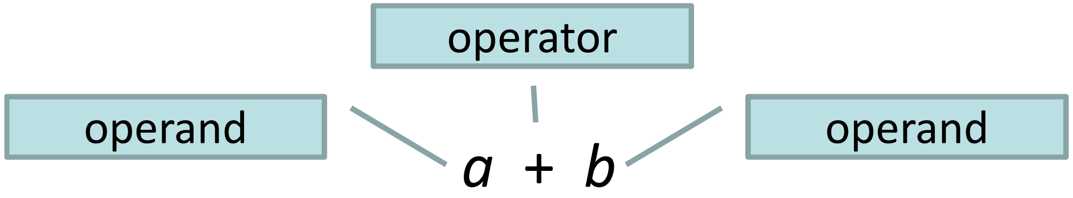
Expressions

An expression is a combination of data by means of one or several operators (e.g., *sum*)

Data can be literal values, variables, constants, and other expressions

> calls to methods can be also included

Data symbols in an expression are called **operands**



Expression composition is guided by rules

For example, operands must have a concrete type to be used in an operation

Non-initialized variables cannot be used in expressions

Compilation error

Operations with data

Arithmetic

Operate with numbers; the result is a number

Relational

Operate with numbers; the result is true/false

Conditional

Operate with true/false; the result is true/false

Bitwise

Operate with the binary representation of integer numbers; the result is a number

Assignment

Perform an operation on an expression and assign the resulting value to a variable

Expressions have a returning value

Returning values have a type

Expressions are said to have type

Two numbers

`+    -    *    /   %`

One number

`++   --`

Increasing / decreasing a variable

They can be used in prefix or suffix, and they have a different precedence

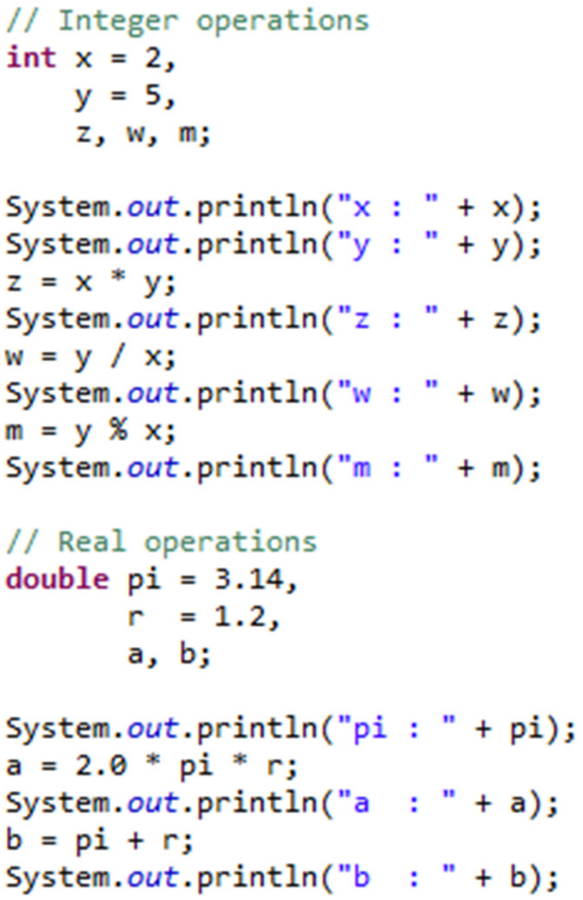
Ej.:

`x++` means increment `x` in 1

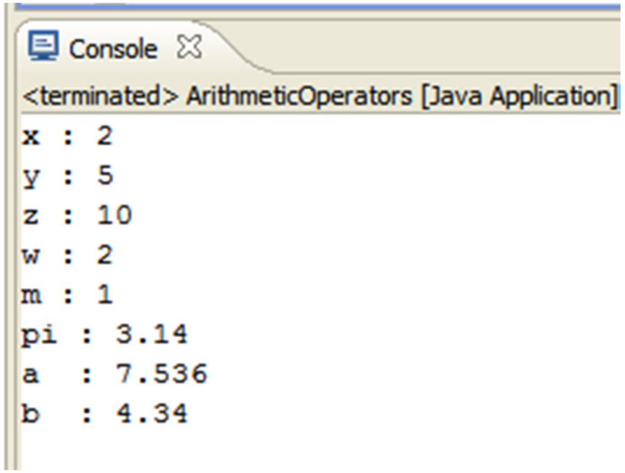
`++y` means increment `y` in 1

5. Operations with data

Arithmetic operators



```
// Integer operations
int x = 2,
    y = 5,
    z, w, m;

System.out.println("x : " + x);
System.out.println("y : " + y);
z = x * y;
System.out.println("z : " + z);
w = y / x;
System.out.println("w : " + w);
m = y % x;
System.out.println("m : " + m);

// Real operations
double pi = 3.14,
       r  = 1.2,
       a, b;

System.out.println("pi : " + pi);
a = 2.0 * pi * r;
System.out.println("a  : " + a);
b = pi + r;
System.out.println("b  : " + b);
```

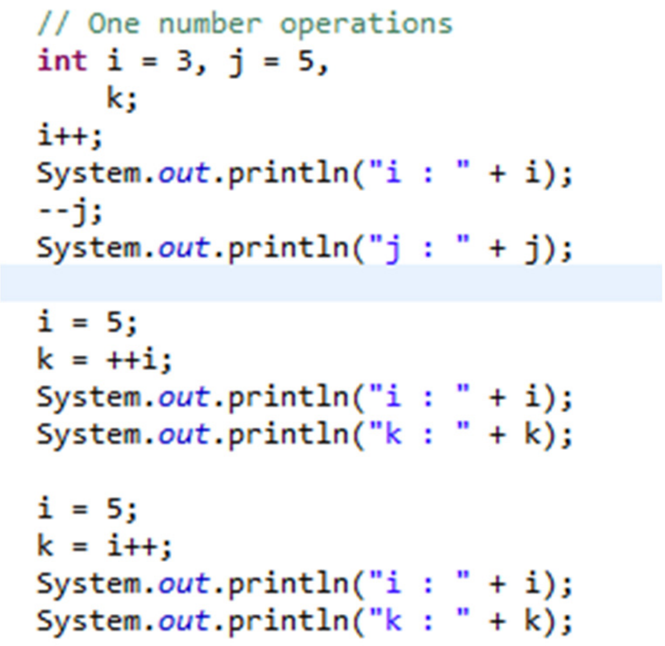


```
<terminated> ArithmeticOperators [Java Application]
x : 2
y : 5
z : 10
w : 2
m : 1
pi : 3.14
a  : 7.536
b  : 4.34
```

ArithmeticOperators.java

5. Operations with data

Arithmetic operators



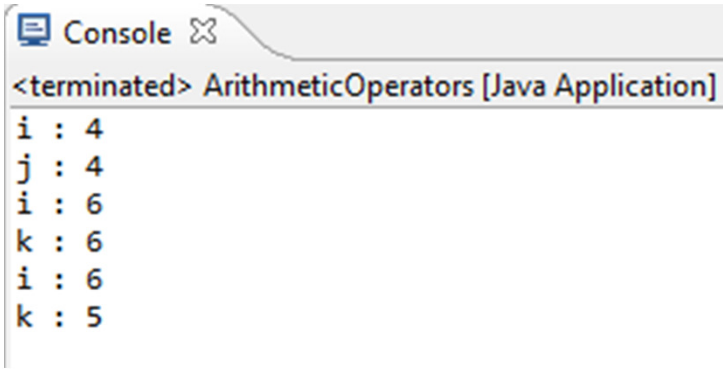
```
// One number operations
int i = 3, j = 5,
    k;
i++;
System.out.println("i : " + i);
--j;
System.out.println("j : " + j);

i = 5;
k = ++i;
System.out.println("i : " + i);
System.out.println("k : " + k);

i = 5;
k = i++;
System.out.println("i : " + i);
System.out.println("k : " + k);
```

k = ++i is equivalent to

```
i = i + 1;
k = i;
```

k = i++ is equivalent to

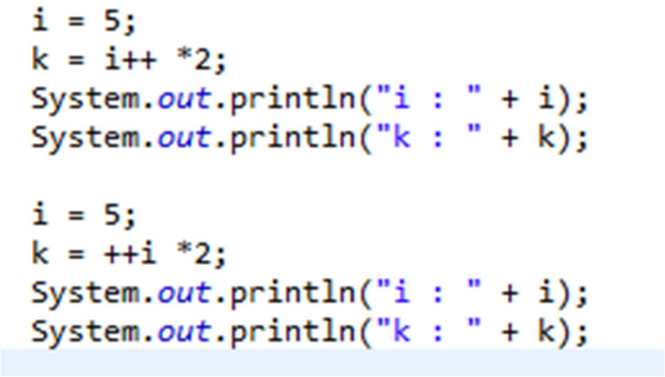
```
k = i;
i = i + 1;
```



```
<terminated> ArithmeticOperators [Java Application]
i : 4
j : 4
i : 6
k : 6
i : 6
k : 5
```

5. Operations with data

Arithmetic operators



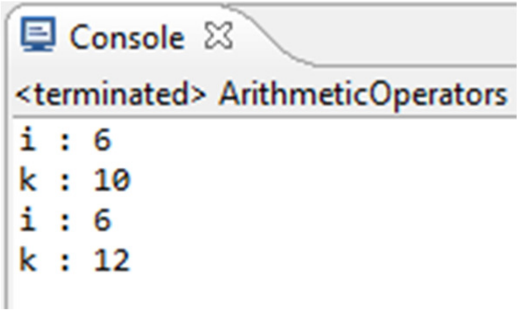
```
i = 5;
k = i++ *2;
System.out.println("i : " + i);
System.out.println("k : " + k);

i = 5;
k = ++i *2;
System.out.println("i : " + i);
System.out.println("k : " + k);
```

k = i++ * 2 is equivalent to

```
k = i * 2;
i = i + 1;
```

k = ++i * 2 is equivalent to

```
i = i + 1;
k = i * 2;
```



```
<terminated> ArithmeticOperators
i : 6
k : 10
i : 6
k : 12
```

ArithmeticOperators.java

Used for comparisons

==   !=   >   <   >=   <=

Have a **boolean** value as result (true/false)

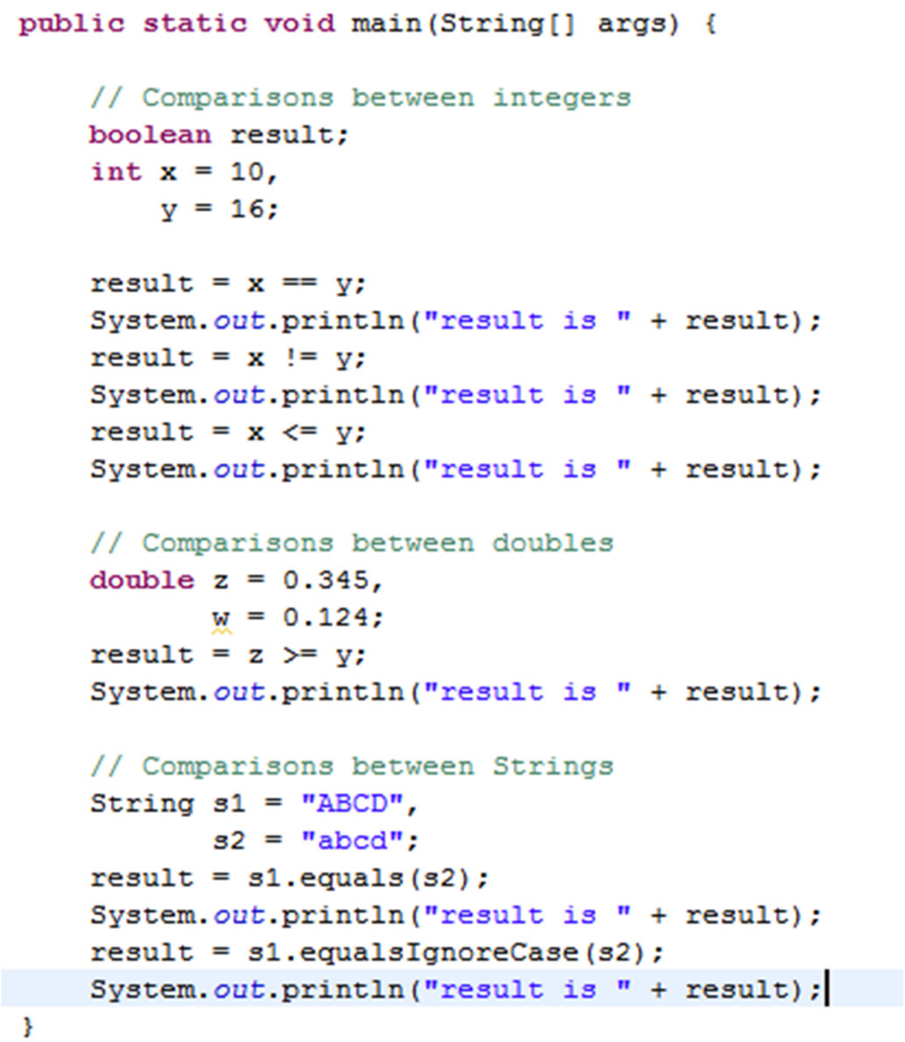
E.g.:

```
boolean result;
int x = 10, y = 16;
result = x == y;   // result is false
result = x <= y;   // result is true
```

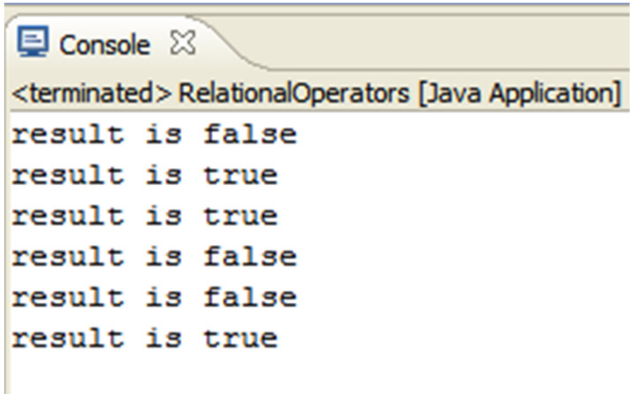
For **String** comparisons, use **equal** method

5. Operations with data

Relational operators



```
public static void main(String[] args) {

    // Comparisons between integers
    boolean result;
    int x = 10,
        y = 16;

    result = x == y;
    System.out.println("result is " + result);
    result = x != y;
    System.out.println("result is " + result);
    result = x <= y;
    System.out.println("result is " + result);

    // Comparisons between doubles
    double z = 0.345,
           w = 0.124;
    result = z >= y;
    System.out.println("result is " + result);

    // Comparisons between Strings
    String s1 = "ABCD",
           s2 = "abcd";
    result = s1.equals(s2);
    System.out.println("result is " + result);
    result = s1.equalsIgnoreCase(s2);
    System.out.println("result is " + result);
}
```



<terminated> RelationalOperators [Java Application]

```
result is false
result is true
result is true
result is false
result is false
result is true
```

RelationalOperators.java

5. Operations with data

Logic operators

Used for operations between **boolean** values

AND: **& &&** OR: **|| |** NOT: **!**

Logic operators are usually **combined with relational operators** to compose complex conditions

Result is a **boolean** value

E.g.:

```
boolean result;
int x = 10, y = 16;
result = (x != 0) &  (x <= y);     // true
result = (x <= y) || (y > 100);    // true
```

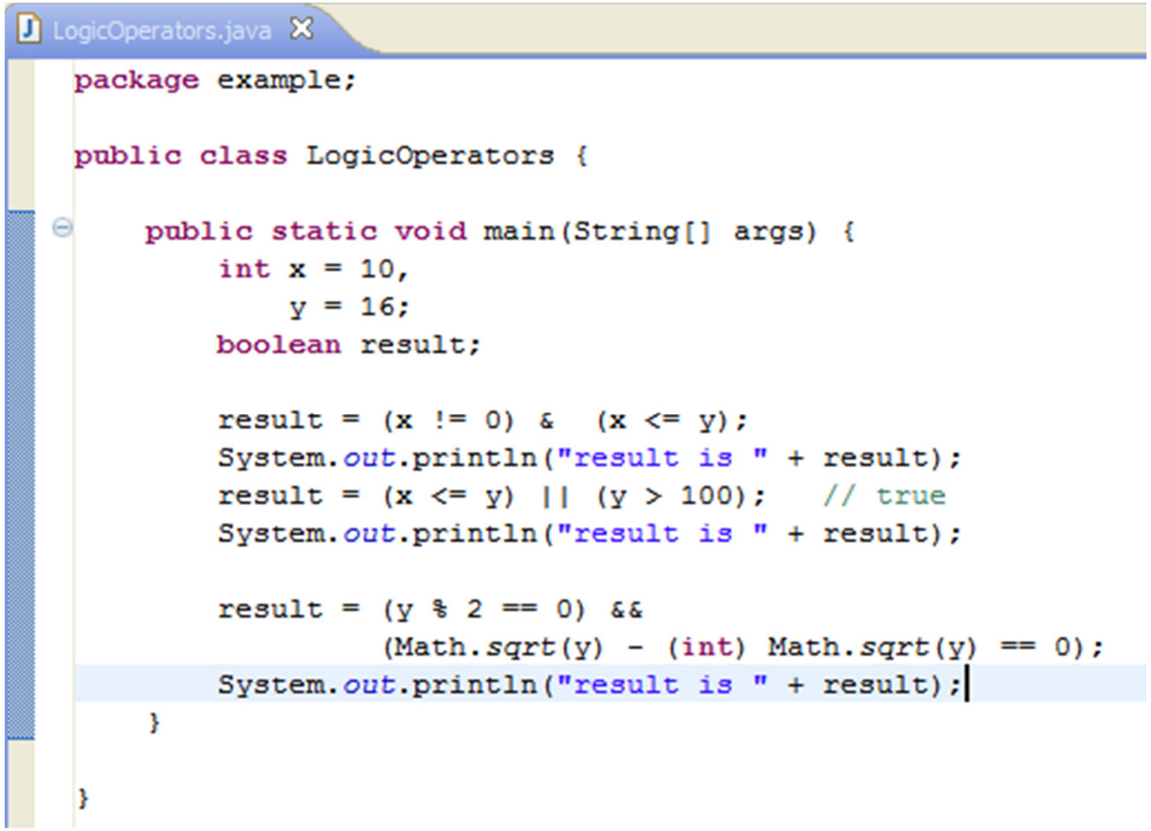
	a AND b		a OR b	
	b is true	*b is false*	*b is true*	*b is false*
a is true	true	false	true	true
a is false	false	false	true	false

| is OR; **||** is OR “short-circuit” (same for **&**, **&&**)

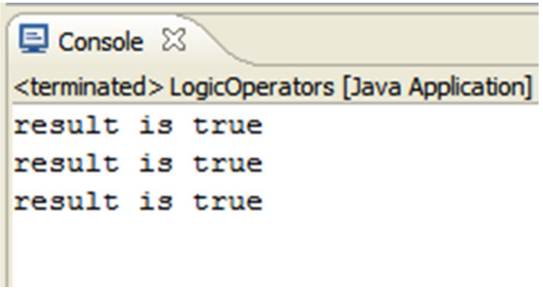
> the evaluation stops when the result is known

5. Operations with data

Logic operators



```java
package example;

public class LogicOperators {

    public static void main(String[] args) {
        int x = 10,
            y = 16;
        boolean result;

        result = (x != 0) &  (x <= y);
        System.out.println("result is " + result);
        result = (x <= y) || (y > 100);   // true
        System.out.println("result is " + result);

        result = (y % 2 == 0) &&
                 (Math.sqrt(y) - (int) Math.sqrt(y) == 0);
        System.out.println("result is " + result);
    }

}
```



```
<terminated> LogicOperators [Java Application]
result is true
result is true
result is true
```

LogicOperators.java

Operations on the bit-based internal representation of integer values

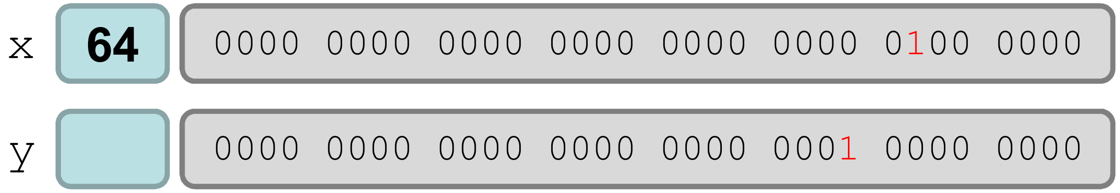
`~`	NOT
`&`	AND
`|`	OR
`^`	XOR
`>>`	SHIFT right
`>>>`	SHIFT right with carry
`<<`	SHIFT left

Have an **`int`** value as result

> **`short`** and **`byte`** are promoted to **`int`**

Ej.:

```
int x = 64;
int y = x << 2;
```

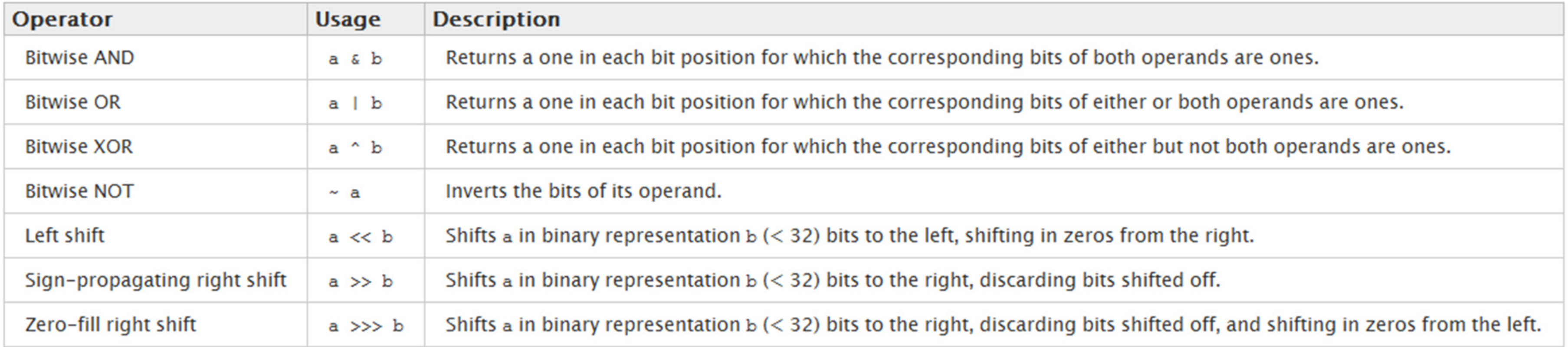


x 64 0000 0000 0000 0000 0000 0000 0100 0000

y 0000 0000 0000 0000 0000 0001 0000 0000

5. Operations with data

Bitwise operators



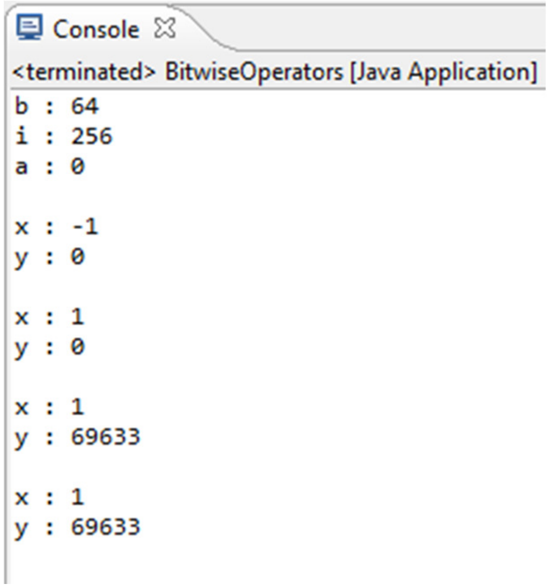
Operator	Usage	Description
Bitwise AND	`a & b`	Returns a one in each bit position for which the corresponding bits of both operands are ones.
Bitwise OR	`a | b`	Returns a one in each bit position for which the corresponding bits of either or both operands are ones.
Bitwise XOR	`a ^ b`	Returns a one in each bit position for which the corresponding bits of either but not both operands are ones.
Bitwise NOT	`~ a`	Inverts the bits of its operand.
Left shift	`a << b`	Shifts `a` in binary representation `b` (< 32) bits to the left, shifting in zeros from the right.
Sign-propagating right shift	`a >> b`	Shifts `a` in binary representation `b` (< 32) bits to the right, discarding bits shifted off.
Zero-fill right shift	`a >>> b`	Shifts `a` in binary representation `b` (< 32) bits to the right, discarding bits shifted off, and shifting in zeros from the left.

Source: mozilla.org

5. Operations with data

Bitwise operators

```
byte b = 64, a;
int i;
i = b << 2;
a = (byte) (b << 2);
System.out.println("b : " + b);
System.out.println("i : " + i);
System.out.println("a : " + a);

int x = 0xFFFFFFFF;
int y = ~x;
System.out.println("\nx : " + x);
System.out.println("y : " + y);

int bitmask = 0x00011000;
x = 0x00000001;
y = x & bitmask ;
System.out.println("\nx : " + x);
System.out.println("y : " + y);

x = 0x00000001;
y = x | bitmask ;
System.out.println("\nx : " + x);
System.out.println("y : " + y);

x = 0x00000001;
y = x ^ bitmask ;
System.out.println("\nx : " + x);
System.out.println("y : " + y);
```



```
<terminated> BitwiseOperators [Java Application]
b : 64
i : 256
a : 0

x : -1
y : 0

x : 1
y : 0

x : 1
y : 69633

x : 1
y : 69633
```

BitwiseOperators.java

5. Operations with data

Assignment operators

Change the value of the variable on the left by the result of the operator applied on the variable and the expression on the right

`<v> <op>= <exp>` is equivalent to `<v> = <v> <op> <exp>`

```
=   +=   -=   *=   /=   %=   &=
|=  ^=   <<=  >>=  >>>=
```

Abbreviation for an operation and a assignment

E.g.:

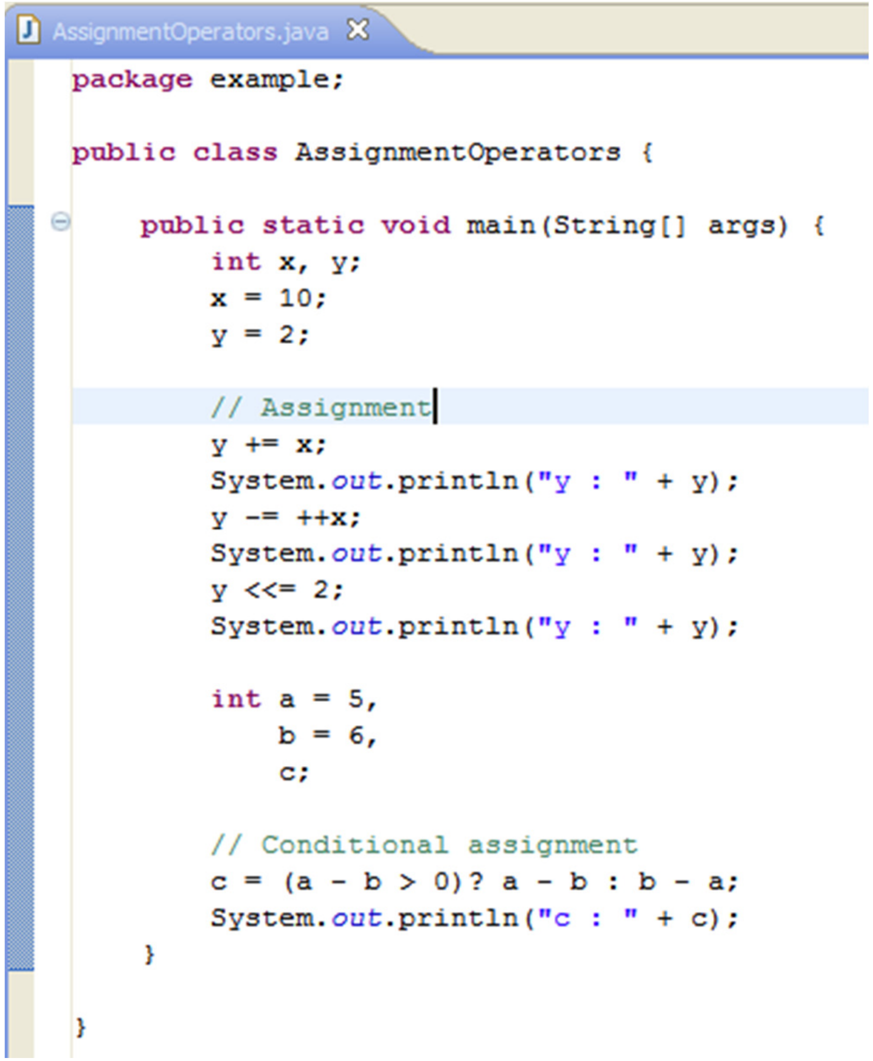
```
int x = 10, y = 2;
y += x;   // y = y + x;        (y : 12)
y -= ++x; // y = y - (x + 1);  (y : -9)
```

Special abbreviation involving boolean values:

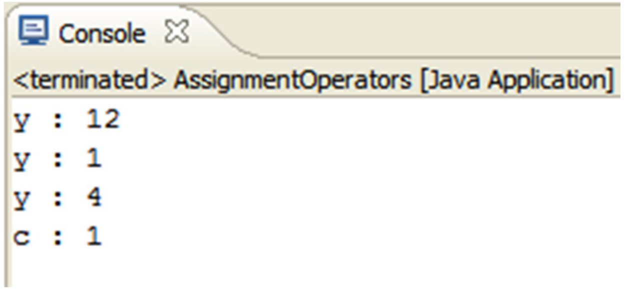
```
<variable> =
    <logical expression> ?
    <value if true> : <value if false>;
```

5. Operations with data

Assignment operators



```
package example;

public class AssignmentOperators {

    public static void main(String[] args) {
        int x, y;
        x = 10;
        y = 2;

        // Assignment
        y += x;
        System.out.println("y : " + y);
        y -= ++x;
        System.out.println("y : " + y);
        y <<= 2;
        System.out.println("y : " + y);

        int a = 5,
            b = 6,
            c;

        // Conditional assignment
        c = (a - b > 0)? a - b : b - a;
        System.out.println("c : " + c);
    }

}
```



```
<terminated> AssignmentOperators [Java Application]
y : 12
y : 1
y : 4
c : 1
```

AssignmentOperators.java

5. Operations with data

Parentheses

Precedence

If not specified, expressions are evaluated in a predefined order

> not directly from left to right

Similar to usual mathematical operator precedence

Parentheses ( ) are used when:

The order of operator application is ambiguous

We want to give higher precedence to some operators over others

We want to make the code more readable / understandable

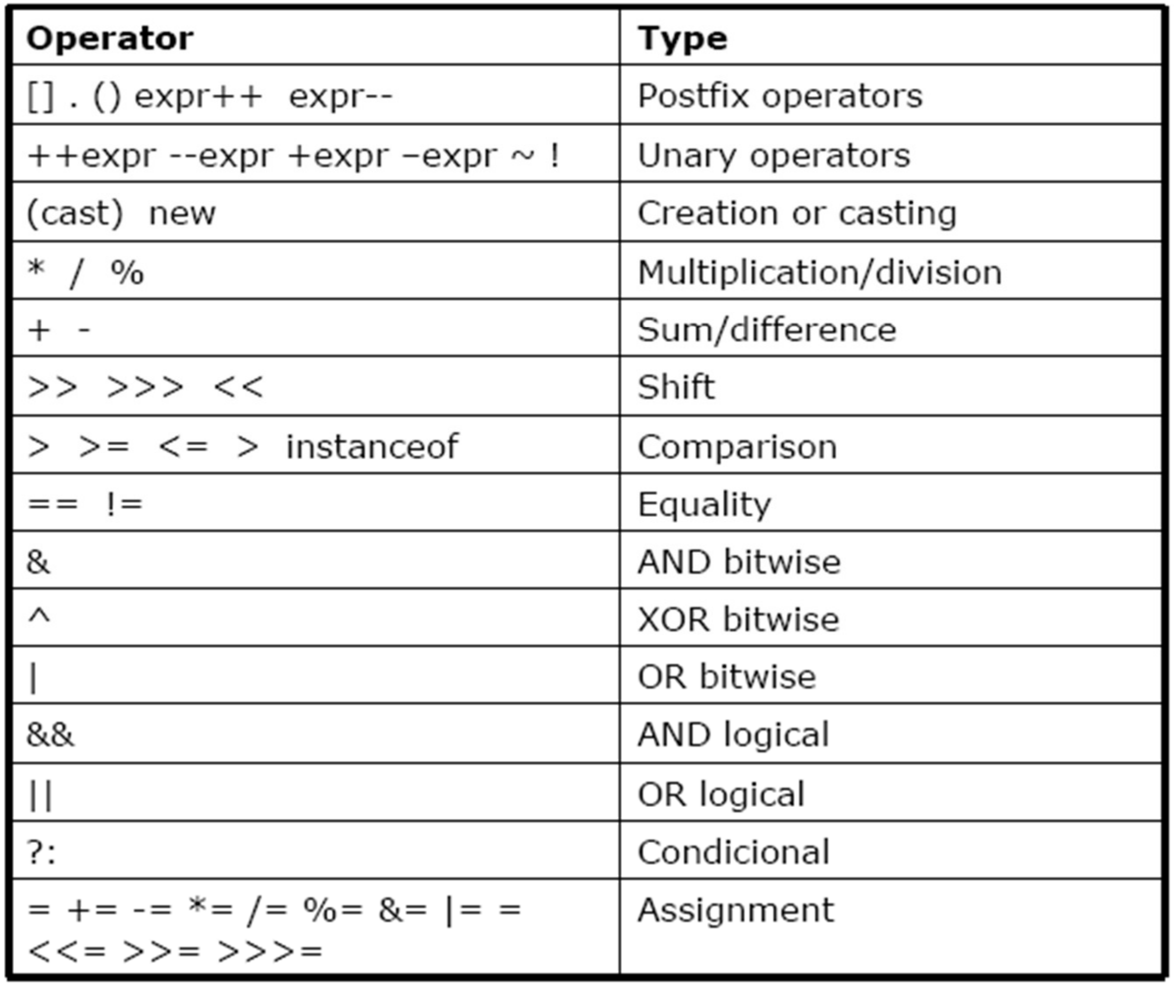
E.g.:

```
int x = 3, y = 4, z = 5;
a =  x + y  * z;              // a : 23
a =  x + (y * z);             // a : 23

a = (x + y) * z;              // a : 35
a = (x * z) + (y * z);        // a : 35
```

5. Operations with data

Precedence

HIGHER
PRECEDENCE



Operator	Type
[] . () expr++ expr--	Postfix operators
++expr --expr +expr −expr ~ !	Unary operators
(cast) new	Creation or casting
* / %	Multiplication/division
+ -	Sum/difference
>> >>> <<	Shift
> >= <= > instanceof	Comparison
== !=	Equality
&	AND bitwise
^	XOR bitwise
|	OR bitwise
&&	AND logical
||	OR logical
?:	Condicional
= += -= *= /= %= &= |= = <<= >>= >>>=	Assignment

LOWER
PRECEDENCE

Outline

1. Basic data types and variables
2. Arrays
3. Input and output
4. Comments
5. Operations with data
6. **Casting between data types**
7. Enumerates
8. Classes as data structures

6. Casting between data types

Automatic promotion and explicit casting

Automatic promotion

Assigning a value of type *A* to a variable of type *B* is only allowed when *A* is "bigger" than *B* (no information is lost in the conversion!)

integers can be assigned to *floats*

```
float <-- int
```

chars can be assigned to *integers*

```
int <-- char
```

Direct assignment, no special code is required

Type casting

The programmer can enforce the conversion in the opposite direction, from a "bigger" type to a "smaller" type (information is lost in the conversion!)

a *float* can be explicitly cast to an *integer*

```
int <-- (int) float
```

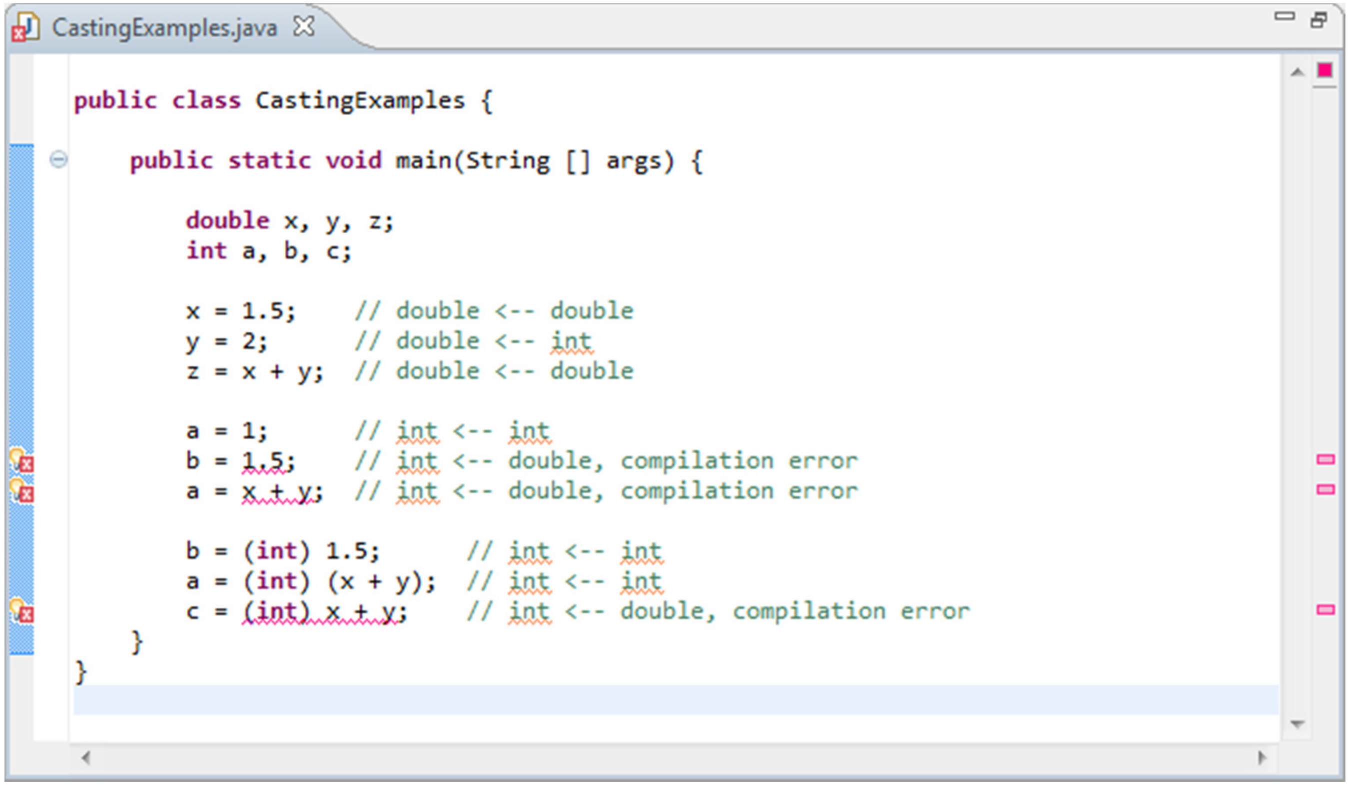
the floating part is removed

Use the explicit casting operator

`(<destination type>)` (besides the expression to cast)

6. Casting between data types

Examples



```
public class CastingExamples {

    public static void main(String [] args) {

        double x, y, z;
        int a, b, c;

        x = 1.5;    // double <-- double
        y = 2;      // double <-- int
        z = x + y;  // double <-- double

        a = 1;      // int <-- int
        b = 1.5;    // int <-- double, compilation error
        a = x + y;  // int <-- double, compilation error

        b = (int) 1.5;      // int <-- int
        a = (int) (x + y);  // int <-- int
        c = (int) x + y;    // int <-- double, compilation error
    }
}
```

CastingExamples.java

Outline

1. Basic data types and variables
2. Arrays
3. Input and output
4. Comments
5. Operations with data
6. Casting between data types
7. **Enumerates**
8. Classes as data structures

7. Enumerates

Definition

New data types can be created by enumeration of the **allowed values of the new type**

> Create a new type named `DayOfTheWeek` with allowed values `{Mon, Tue, Wed, Thu, Fri, Sat, Sun}`

New variables with type `DayOfTheWeek` can be created

These variables can store the values defined in the enumerate

Syntax

Definition

```
enum <type identifier> {<value 1>, …, <value n>};
```

E.g.:

```
enum DayOfTheWeek {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
```

`enum` declarations must be **outside of the main procedure!**

Use

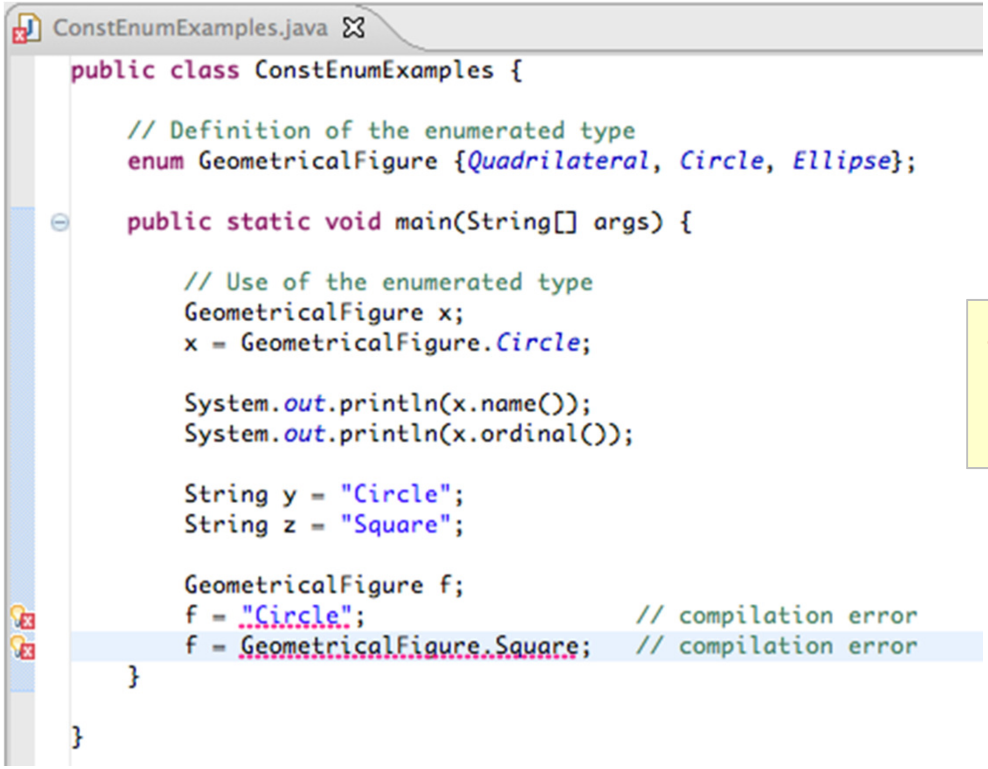
E.g.:

```
DayOfTheWeek x;
x = DayOfTheWeek.Mon;
```

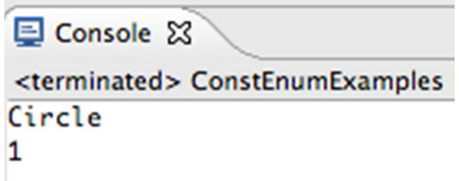
7. Enumerates

Definition

CostEnumExamples.java



```
public class ConstEnumExamples {

    // Definition of the enumerated type
    enum GeometricalFigure {Quadrilateral, Circle, Ellipse};

    public static void main(String[] args) {

        // Use of the enumerated type
        GeometricalFigure x;
        x = GeometricalFigure.Circle;

        System.out.println(x.name());
        System.out.println(x.ordinal());

        String y = "Circle";
        String z = "Square";

        GeometricalFigure f;
        f = "Circle";                    // compilation error
        f = GeometricalFigure.Square;    // compilation error
    }

}
```



```
<terminated> ConstEnumExamples
Circle
1
```

enums are similar to `Strings`
but `enums` restrict the possible values of the "string"

Outline

1. Basic data types and variables
2. Arrays
3. Input and output
4. Comments
5. Operations with data
6. Casting between data types
7. Enumerates
8. **Classes as data structures**

8. Classes as data structures

Concept

An **object** can be seen as a **data structure** that represents an entity of the domain

Object Entry *#5* of an address book

“Juan”

“Gomez Romero”

29

“jgromero@inf.uc3m.es”

Object 2D point *p*

(2.1, 3.2)

> ***collection of values of different types*** which are managed together

An object belongs to a **class**, where the **attributes** or fields of the objects of the class are defined

Class Entry of an address book

> Name

> Surname

> Age

> E-mail

Class 2D point

> x coordinate

> y coordinate

Programmers can define classes their own classes and use objects in their applications

Class definition

```
[modifiers] class <name of the class> {
     <attributes>
}
```

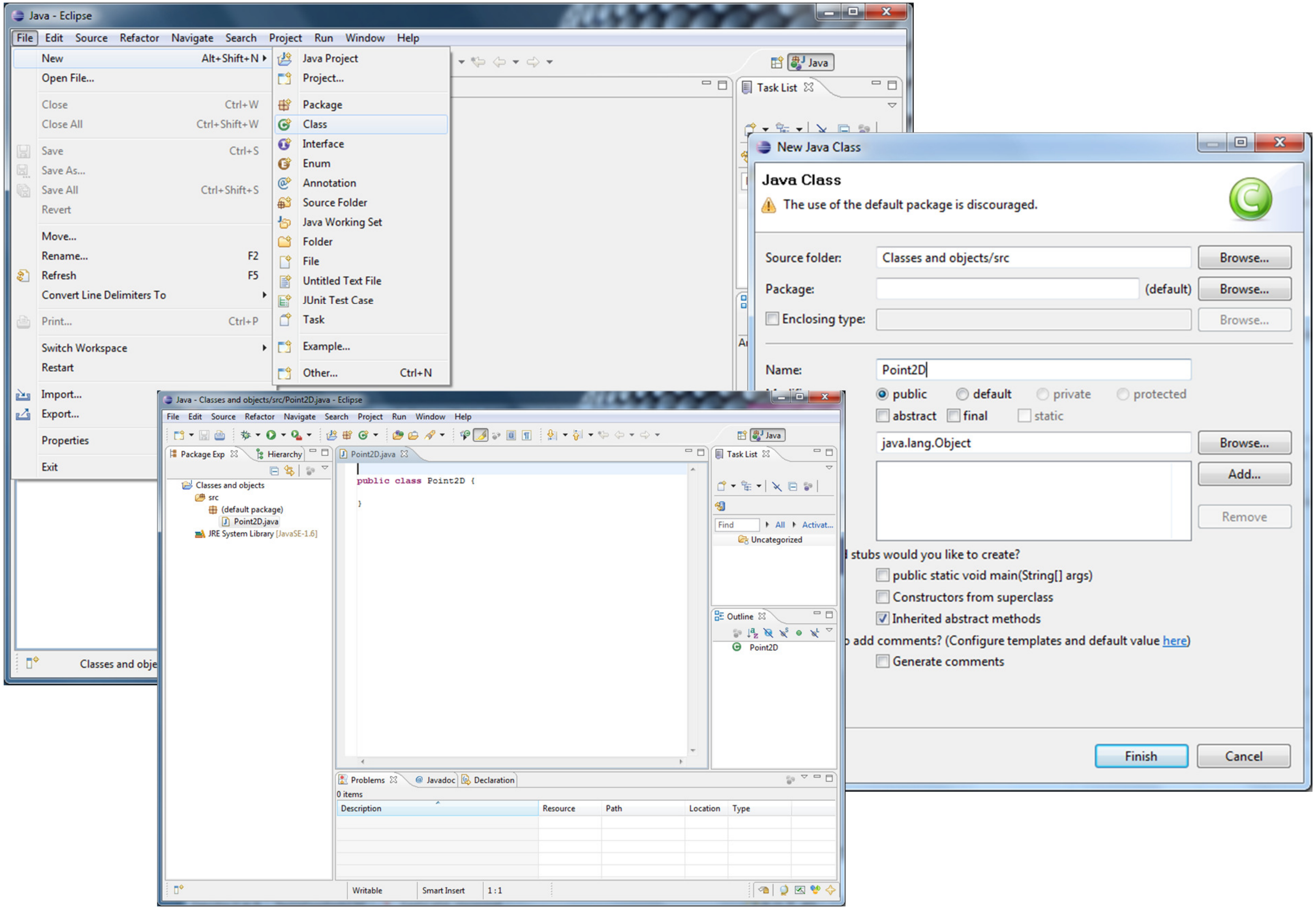
```
[modifiers]
```

`public`	**The class can be used by any other class**
`abstract`	Objects cannot be created for this class, but subclasses are allowed
`final`	Subclasses are not allowed
`none`	By default, the class can be used by the classes of the same package

```
<name of the class>
   valid Java identifier
```

E.g.: **`public class Point2D { … }`**

8. Classes as data structures



New Java Class

Java Class

⚠ The use of the default package is discouraged.

Source folder: Classes and objects/src

Package: (default)

□ Enclosing type:

Name: Point2D

● public ○ default ○ private ○ protected

□ abstract □ final □ static

java.lang.Object

stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

add comments? (Configure templates and default value here)

□ Generate comments

```
public class Point2D {

}
```

8. Classes as data structures

Conventions for classes

Only a single `public` class is allowed within a file. That file should be named after the public class that contains with the extension ".java"

Usually, an application consists of numerous .java files

Compilation (`javac`) converts each class definition (.java) into bytecode (.class)

The execution of the application starts from the class that contains the `main()`

Several classes can be grouped in packages, in the same way as classes of the Java platform

8. Classes as data structures

Class attributes definition

A class defines the attributes (or fields) of the objects that belong to (or are *members of*) the class

Attributes definition

Syntax

```
[modifiers] <type> <name of the attribute>;
```

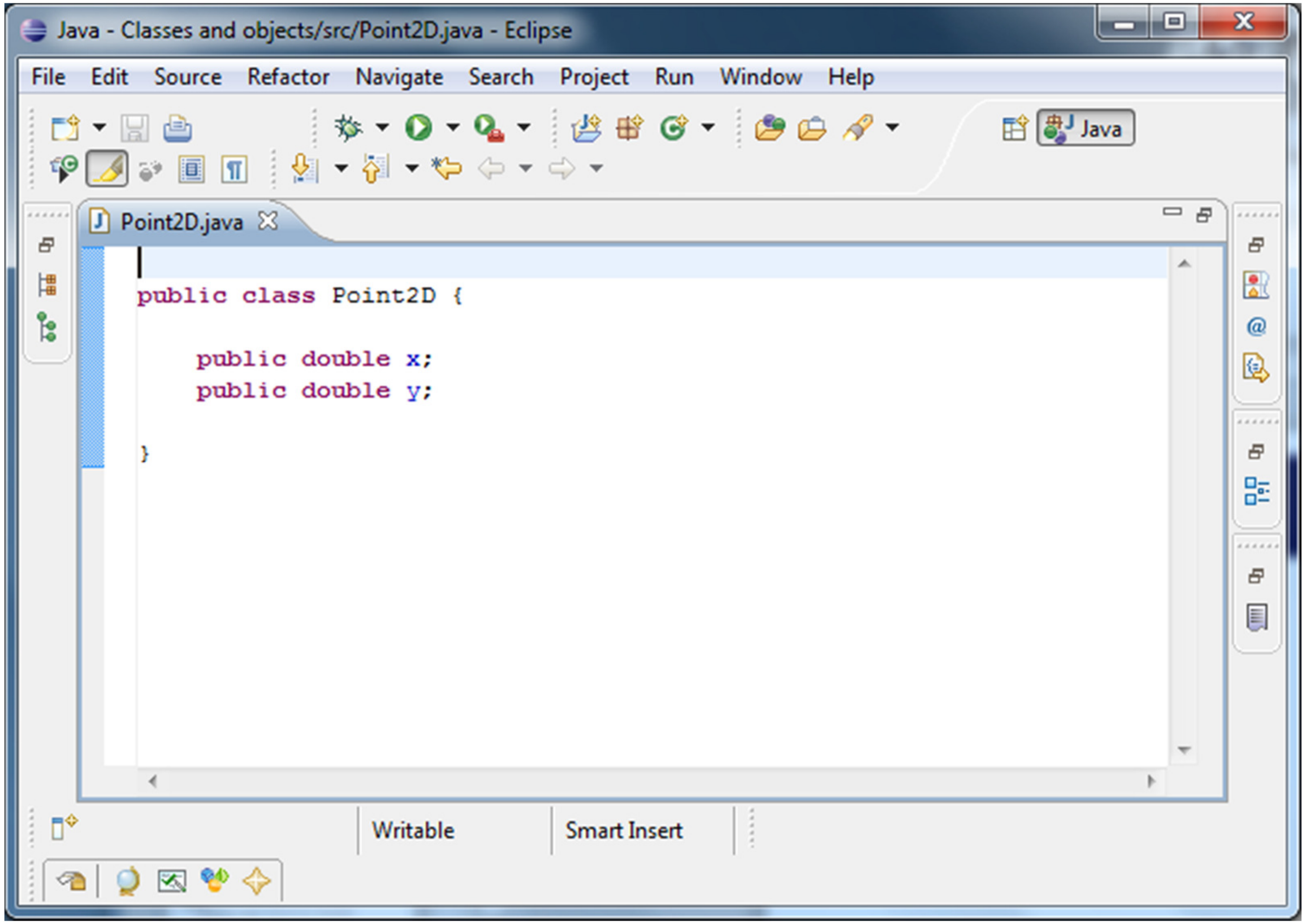
```
[modifiers]
```

`public`	**The attribute can be accessed from any other class**
`private`	The attribute cannot be accessed from any class other than this
`protected`	The attribute can be accessed only from this class and its subclasses
`package`	The attribute can be accessed from any other class inside this package

E.g.: **`public double x;`**

8. Classes as data structures

Example

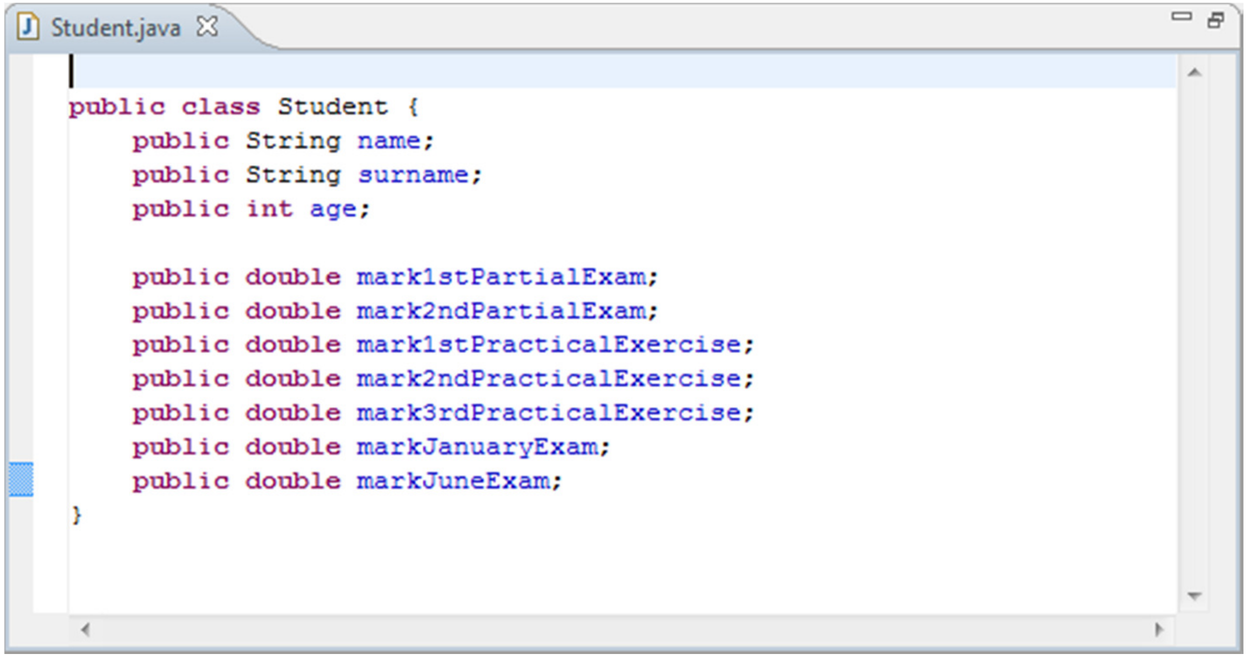


Java - Classes and objects/src/Point2D.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Point2D.java
Writable
Smart Insert

```java

public class Point2D {

    public double x;
    public double y;

}
```

8. Classes as data structures

Example



```java

public class Student {
    public String name;
    public String surname;
    public int age;

    public double mark1stPartialExam;
    public double mark2ndPartialExam;
    public double mark1stPracticalExercise;
    public double mark2ndPracticalExercise;
    public double mark3rdPracticalExercise;
    public double markJanuaryExam;
    public double markJuneExam;
}
```

Classes are not directly used

Instead, once classes have been implemented,

1. Create a class with a `main` method

   > the program begins here

2. Inside the `main`,
   1. Declare object variables
   2. Create objects (allocate memory for an object instance)
   3. Operate with objects

1. Declare object variables

Object variables are declared as basic data type variables

An object declaration declares a reference to the object, not the object itself

Basic syntax

```
<class name> <variable name>;
```

E.g.:

```
Point2D p1;
Student stud;
```

2. Create objects (allocate memory)

Operator **new** creates a new object of a class (memory for the object is allocated)

This object is assigned to a reference (variable previously defined) of the type of the class

Basic syntax

```
<variable name> = <new> <class name>();
```

E.g.:

```
p1 = new Point2D();
st = new Student();
```

3. Operate with objects

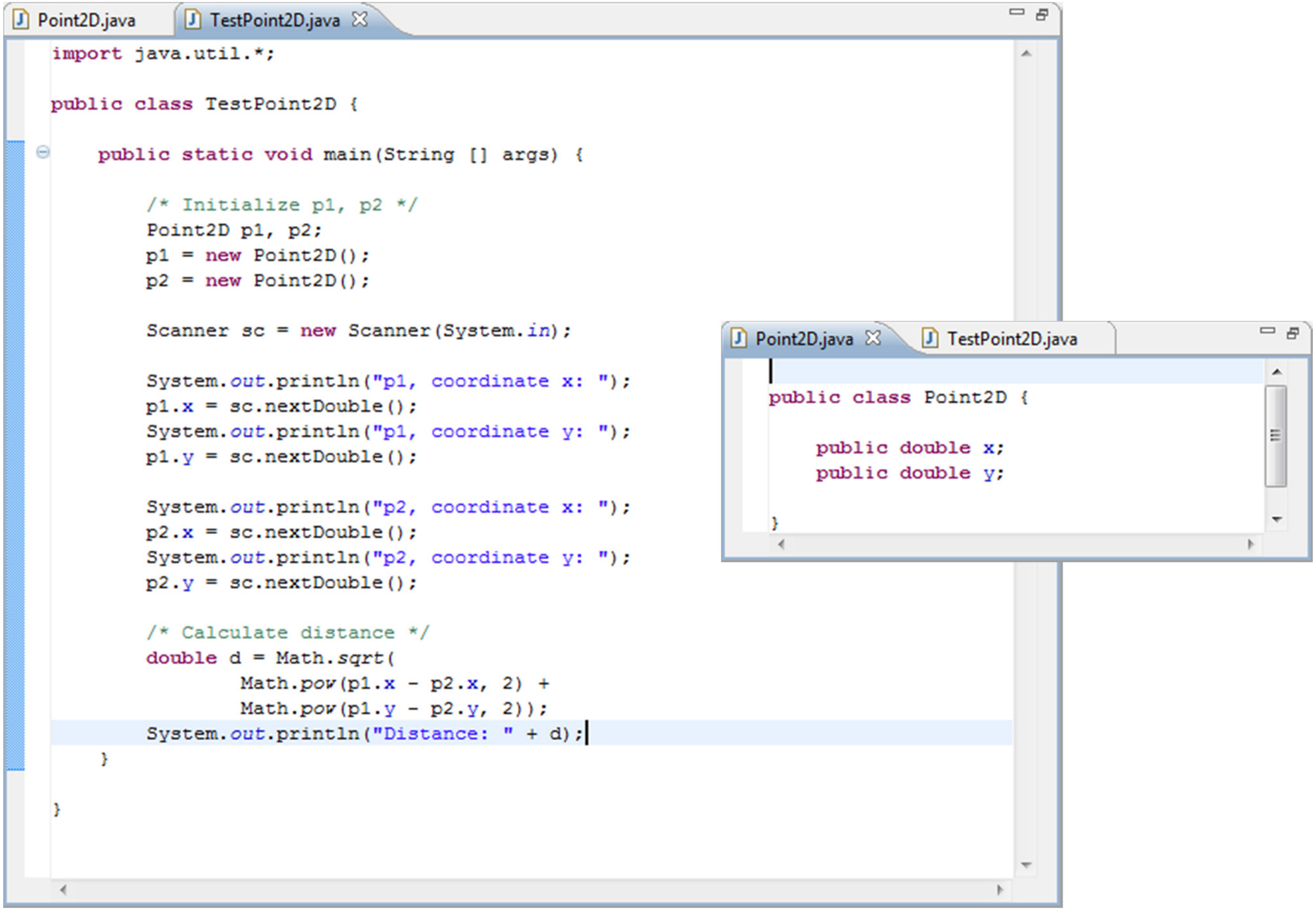
Use the dot operator (.) to access to object attributes

> attributes can be seen as a collection of variables grouped in the object

E.g.:

```
p1.x = 2.1;
p1.y = 3.2;
System.out.println(
    "Position (" + p1.x + ", " + p1.y + ")" );
```

8. Classes as data structures



```java
import java.util.*;

public class TestPoint2D {

    public static void main(String [] args) {

        /* Initialize p1, p2 */
        Point2D p1, p2;
        p1 = new Point2D();
        p2 = new Point2D();

        Scanner sc = new Scanner(System.in);

        System.out.println("p1, coordinate x: ");
        p1.x = sc.nextDouble();
        System.out.println("p1, coordinate y: ");
        p1.y = sc.nextDouble();

        System.out.println("p2, coordinate x: ");
        p2.x = sc.nextDouble();
        System.out.println("p2, coordinate y: ");
        p2.y = sc.nextDouble();

        /* Calculate distance */
        double d = Math.sqrt(
                Math.pow(p1.x - p2.x, 2) +
                Math.pow(p1.y - p2.y, 2));
        System.out.println("Distance: " + d);
    }

}
```

```java

public class Point2D {

    public double x;
    public double y;

}
```

8. Classes as data structures

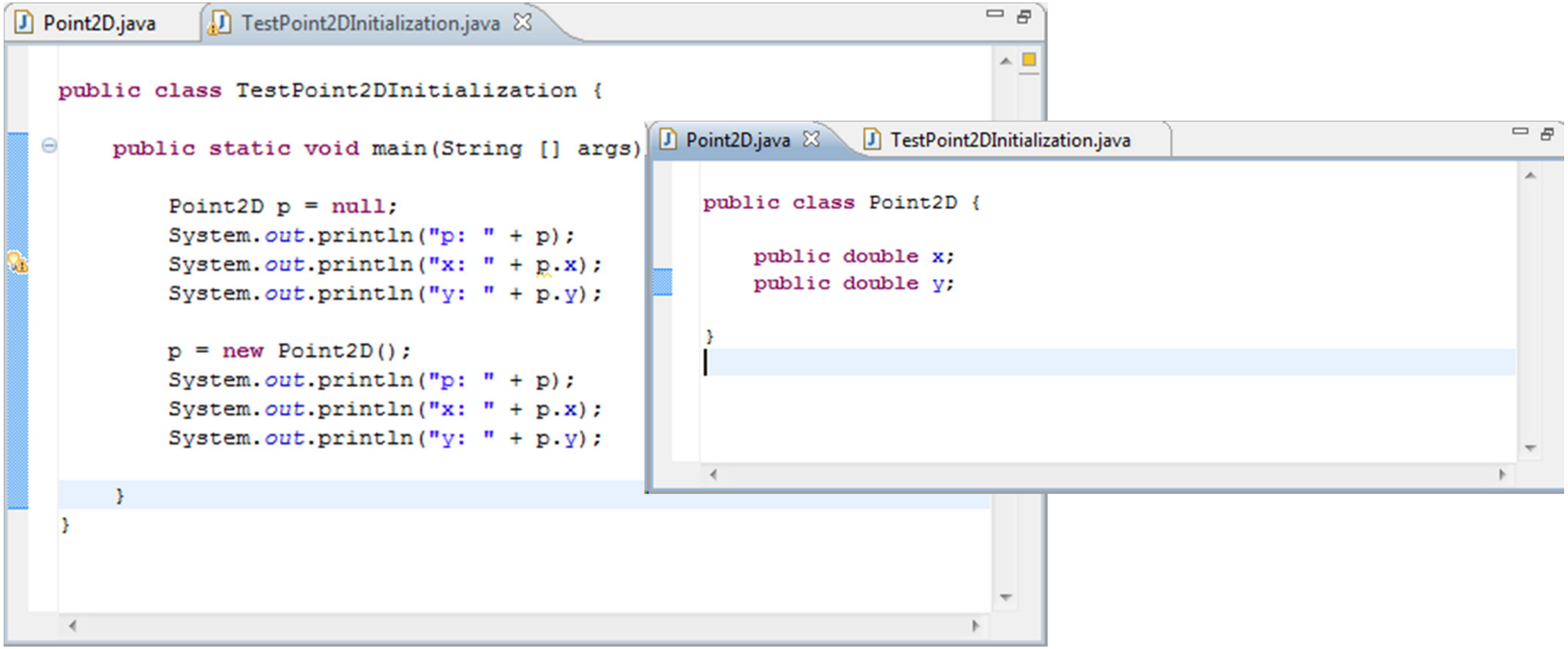
Non-initialized object references and object attributes

- **Object references initial value**
  - The value of an object reference may be the special value **`null`**
  - **`null`** means 'not a valid reference' and can be also used for arrays and `String`
  - If we try to access to the attributes of a null reference, we get a runtime error (`NullPointerException`)

- **Object attributes initial value**
  - The attributes of an object have a default value after creation with **`new`** (0 for integers, false for `boolean`, **`null`** for `String`, etc.) –in the same way as arrays
  - An initial value (other than the default) can be assigned to object attributes in the class declaration
  - Until changed, this is the value of the attributes of any object of the class

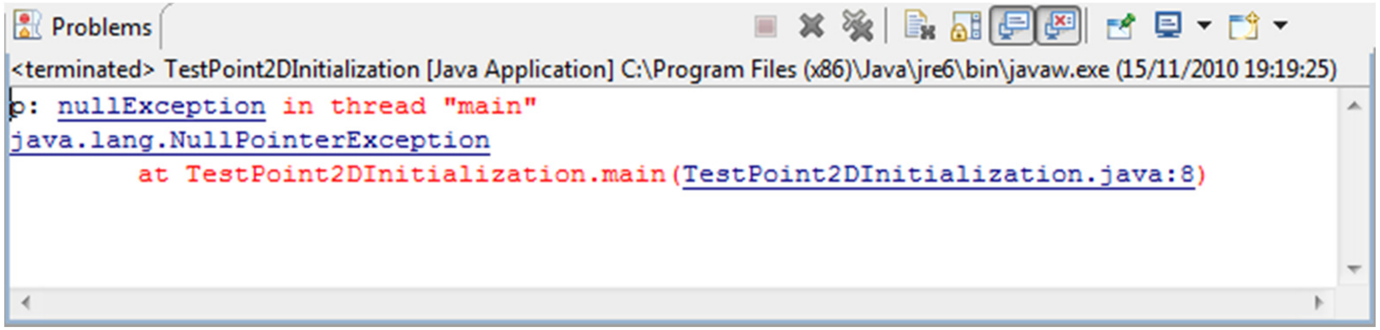
8. Classes as data structures

Example



```java
public class TestPoint2DInitialization {

    public static void main(String [] args) {

        Point2D p = null;
        System.out.println("p: " + p);
        System.out.println("x: " + p.x);
        System.out.println("y: " + p.y);

        p = new Point2D();
        System.out.println("p: " + p);
        System.out.println("x: " + p.x);
        System.out.println("y: " + p.y);

    }
}
```

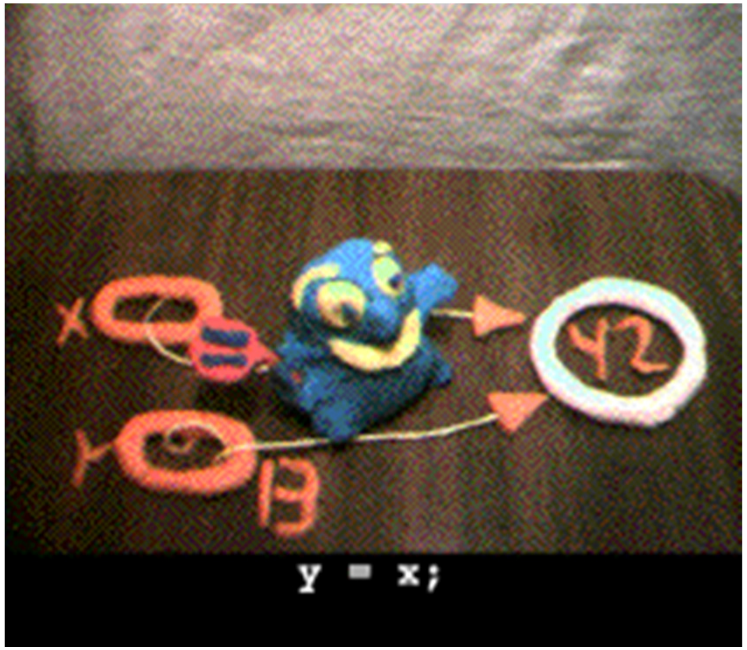
```java
public class Point2D {

    public double x;
    public double y;

}
```



```
<terminated> TestPoint2DInitialization [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (15/11/2010 19:19:25)
p: nullException in thread "main"
java.lang.NullPointerException
	at TestPoint2DInitialization.main(TestPoint2DInitialization.java:8)
```

Pointer fun! http://cslibrary.stanford.edu/104/

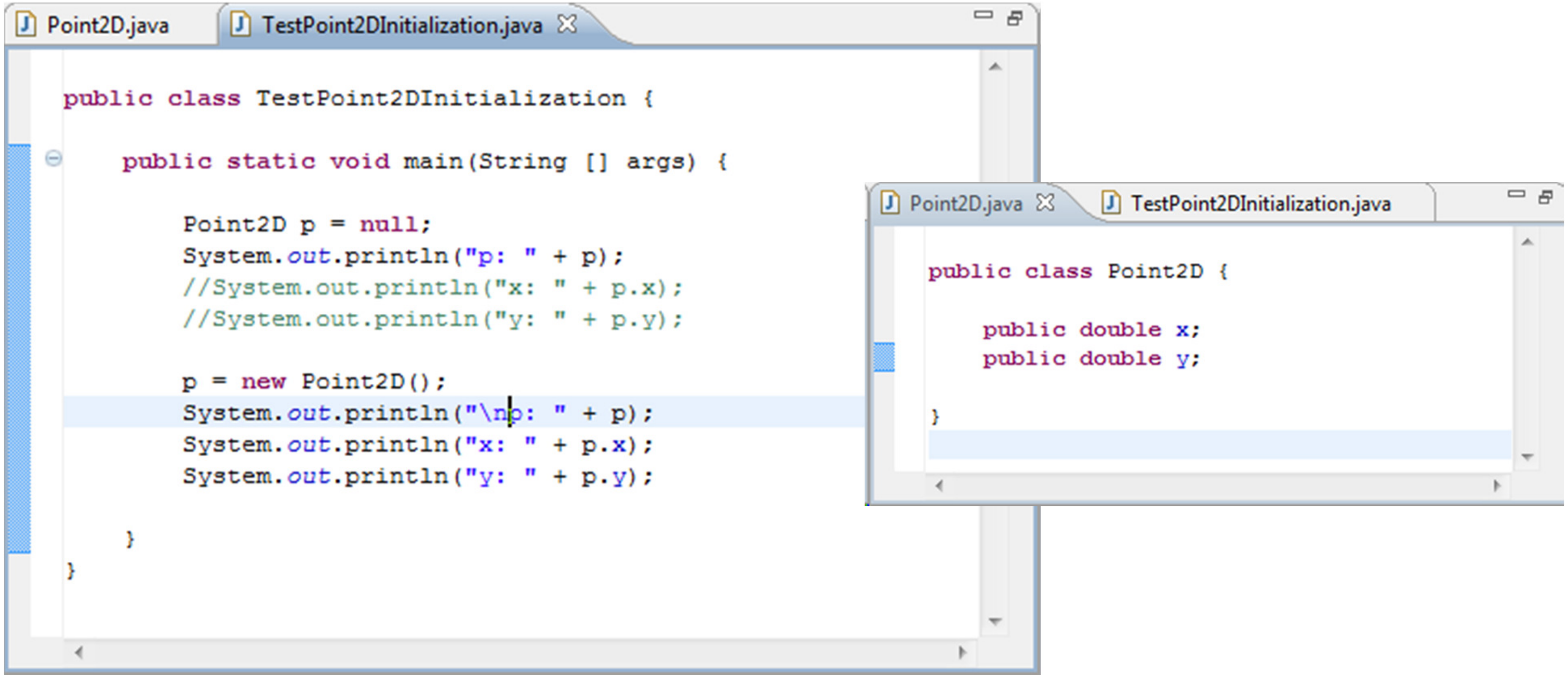
(http://youtu.be/vm5MNP7pn5g)



pointer: reference
pointee: referenced object
dereference: access to referenced object

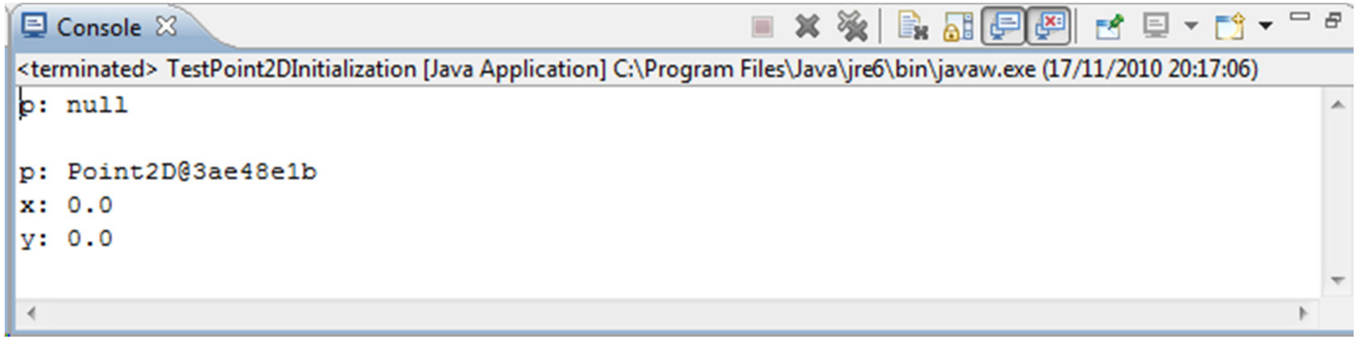
8. Classes as data structures

Example



```java
public class TestPoint2DInitialization {

    public static void main(String [] args) {

        Point2D p = null;
        System.out.println("p: " + p);
        //System.out.println("x: " + p.x);
        //System.out.println("y: " + p.y);

        p = new Point2D();
        System.out.println("\np: " + p);
        System.out.println("x: " + p.x);
        System.out.println("y: " + p.y);

    }
}
```

```java
public class Point2D {

    public double x;
    public double y;

}
```

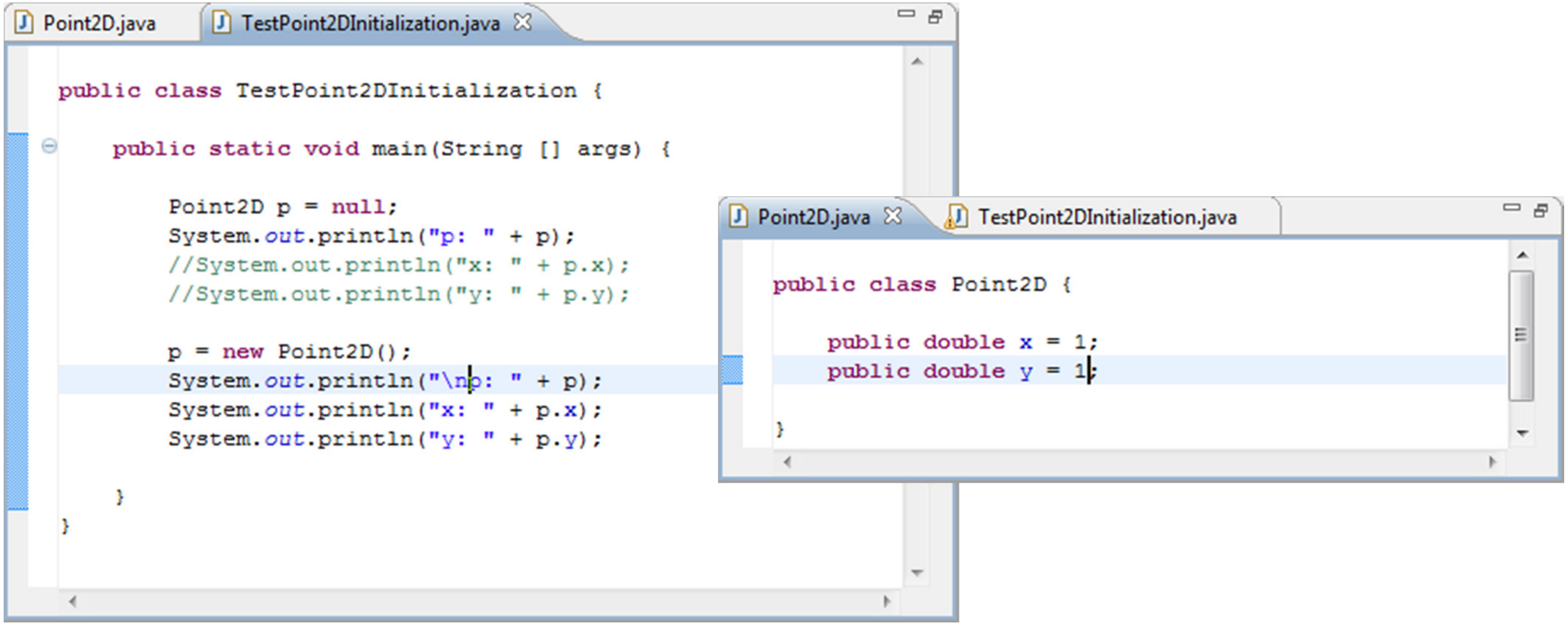


```
<terminated> TestPoint2DInitialization [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (17/11/2010 20:17:06)
p: null

p: Point2D@3ae48e1b
x: 0.0
y: 0.0
```

Universidad
Carlos III de Madrid
www.uc3m.es

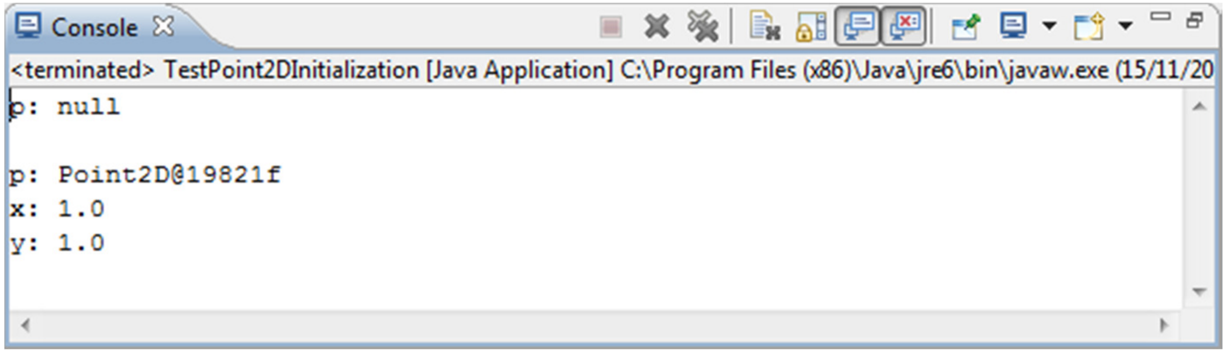
8. Classes as data structures

Example



```java
public class TestPoint2DInitialization {

    public static void main(String [] args) {

        Point2D p = null;
        System.out.println("p: " + p);
        //System.out.println("x: " + p.x);
        //System.out.println("y: " + p.y);

        p = new Point2D();
        System.out.println("\np: " + p);
        System.out.println("x: " + p.x);
        System.out.println("y: " + p.y);

    }
}
```

```java
public class Point2D {

    public double x = 1;
    public double y = 1;

}
```



<terminated> TestPoint2DInitialization [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (15/11/20

```
p: null

p: Point2D@19821f
x: 1.0
y: 1.0
```

Object assignment

Direct object assignment is similar to direct array assignment

(An object variable is a reference to the section of the memory where the object attributes are actually stored.)

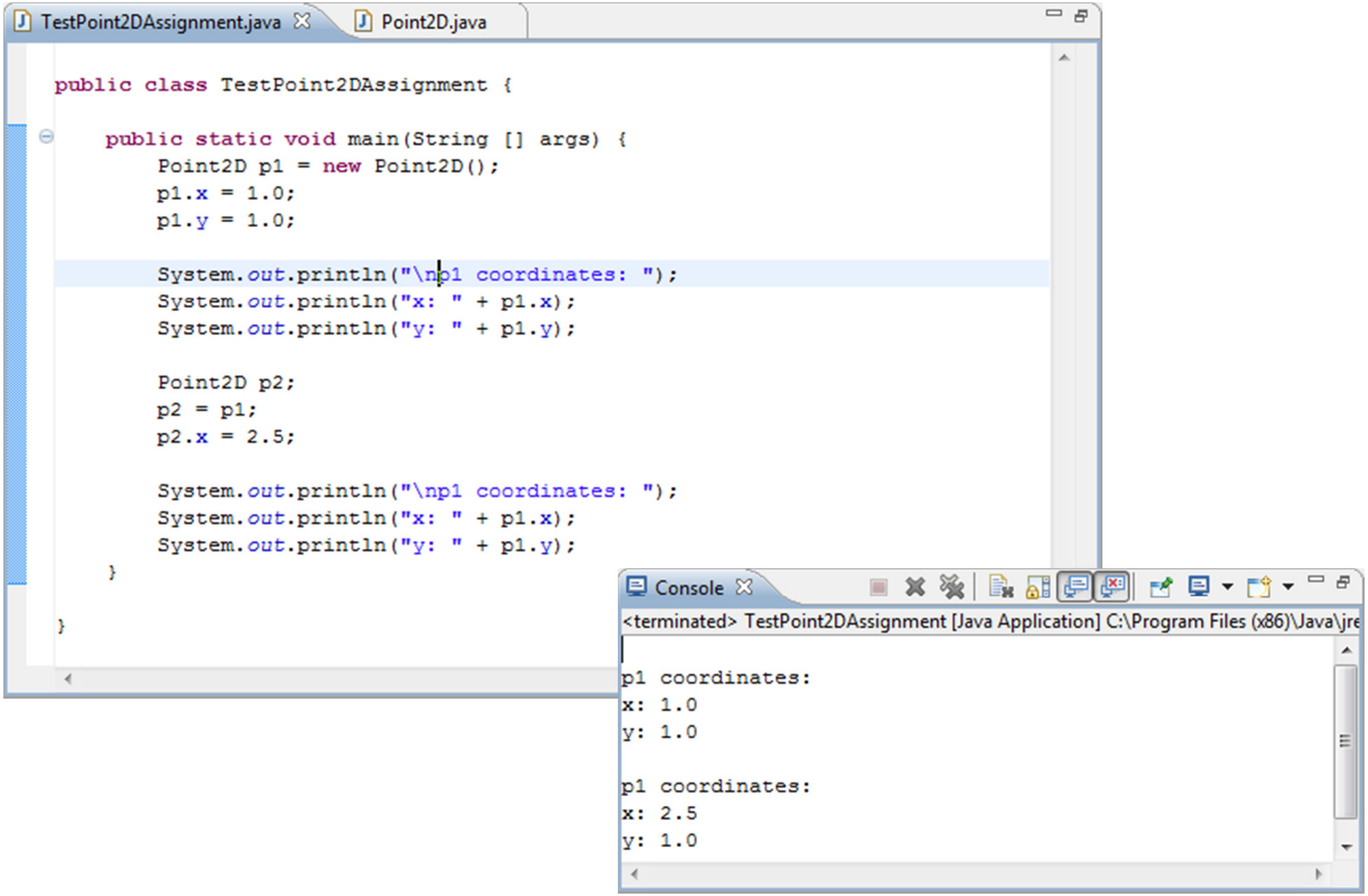
> If two objects are directly assigned, they *point* to the same section of the memory, and consequently, to the same object

> Changes in one reference affect the other reference

> Object copy must be performed attribute by attribute

8. Classes as data structures

Direct assignment



```
public class TestPoint2DAssignment {

    public static void main(String [] args) {
        Point2D p1 = new Point2D();
        p1.x = 1.0;
        p1.y = 1.0;

        System.out.println("\np1 coordinates: ");
        System.out.println("x: " + p1.x);
        System.out.println("y: " + p1.y);

        Point2D p2;
        p2 = p1;
        p2.x = 2.5;

        System.out.println("\np1 coordinates: ");
        System.out.println("x: " + p1.x);
        System.out.println("y: " + p1.y);
    }

}
```

```
<terminated> TestPoint2DAssignment [Java Application] C:\Program Files (x86)\Java\jre

p1 coordinates:
x: 1.0
y: 1.0

p1 coordinates:
x: 2.5
y: 1.0
```

Outline

1. **Basic data types and variables**
2. **Input and output**
3. **Comments**
4. **Arrays**
5. **Operations with data**
6. **Casting between data types**
7. **Enumerates**
8. **Classes as data structures**

Summary

Data and operators

Data in Java

- Basic
  - integers (`int`, `long`, `short`), real (`float`, `double`), character (`char`), boolean (`boolean`), strings (`String`)
- Complex
  - arrays (`[]`)

Variables are used to store values

Variable type is assigned in the variable declaration

Printing (`System.out`) and reading (`Scanner`)

Summary

Data and operators

Operators (arithmetic, relational, logical, bitwise, assignment)

Use of parenthesis when precedence is not clear or the code is confusing

In assignments, the type of the variable and the type of the expression must be compatible

- Explicit casting may be convenient in some cases

Beware of direct assignment of arrays and objects

Use of comments in the code is fundamental

Programmers can define their own data types

- Enumerators
- Classes

Recommended lectures

The Java^{TM} Tutorials. Oracle, **Language Basics** [link]

H. M. Deitel, P. J. Deitel. *Java: How to Program. Prentice Hall, 2007 (7th Edition)*, **Chapters 7** [link]**, L** [link]**, 3** [link]**,**

K. Sierra, B. Bates. *Head First Java.* O'Reilly Media, 2005 (2nd Edition), **Chapter 3** [link]

I. Horton. *Beginning Java 2, JDK 5 Edition*. Wrox, 2004 (5th Edition), **Chapters 2** [link]**, 4** [link]

B. Eckel. *Thinking in Java*. Prentice Hall, 2002 (3rd Edition), **Chapters 1-3** [link]

Programming – Grado en Ingeniería Informática

Authors

Of this version:
Juan Gómez Romero

Based on the work by:
Ángel García Olaya
Manuel Pereira González
Silvia de Castro García
Gustavo Fernández-Baillo Cañas

Universidad
Carlos III de Madrid