OBJECT-ORIENTED PROGRAMMING

1.2 JAVA BASICS

Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn

1

Content

2

1. Identifiers

- Identifiers:
  - A set of characters representing variables, methods, classes and labels
- Naming rules:
  - Characters can be numbers, alphabets, '$' or '_'
  - Name must not:
    - Start by a Number
    - Be the same as a keyword
  - Distinguish between UpperCases and LowerCases
    - Yourname, yourname, YourName and yourName are for different identifiers

✓
```
An_Identifier
a_2nd_Identifier
Go2
$10
```

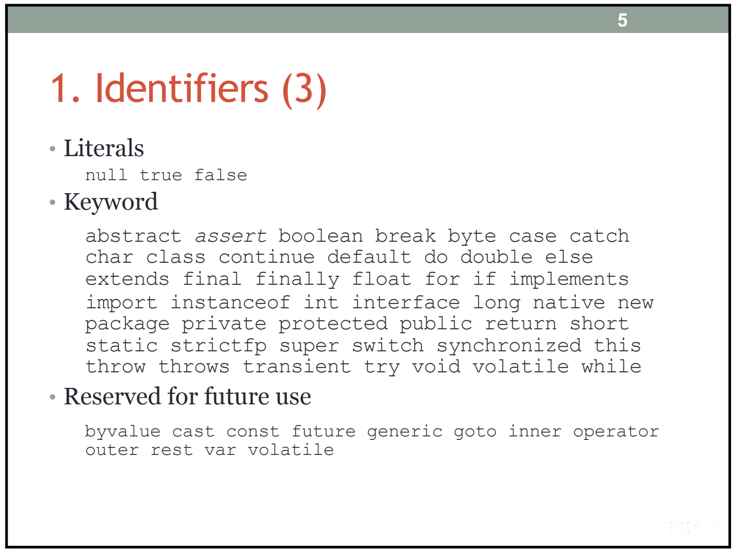
✗
```
An-Identifier
2nd_Identifier
goto
10$
```

3

1. Identifiers (2)

- Naming convention:
  - Start with an Alphabet
  - Package: all in lowercase
    - theexample
  - Class: the first letter of word is in uppercase
    - TheExample
  - Method/field: start with a lowercase letter, the first letter of remaining word is in uppercase
    - theExample
  - Constants: All in uppercase
    - THE_EXAMPLE

4



1. Identifiers (3)

- Literals

```
null true false
```

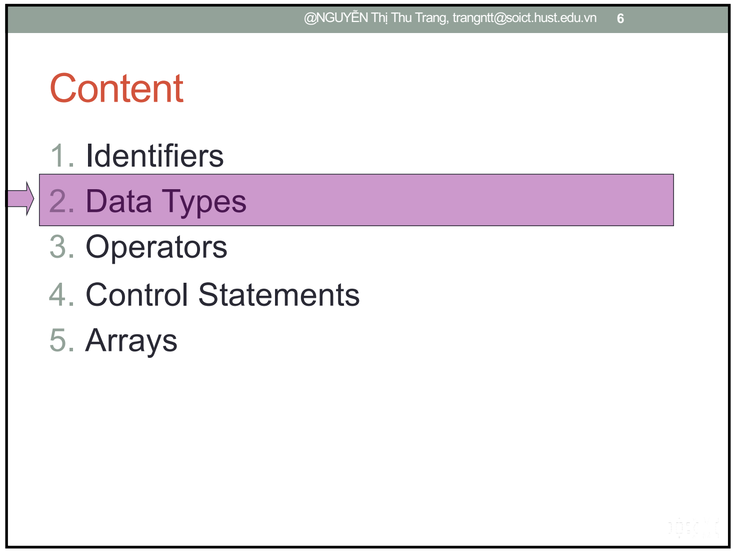
- Keyword

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

- Reserved for future use

```
byvalue cast const future generic goto inner operator
outer rest var volatile
```

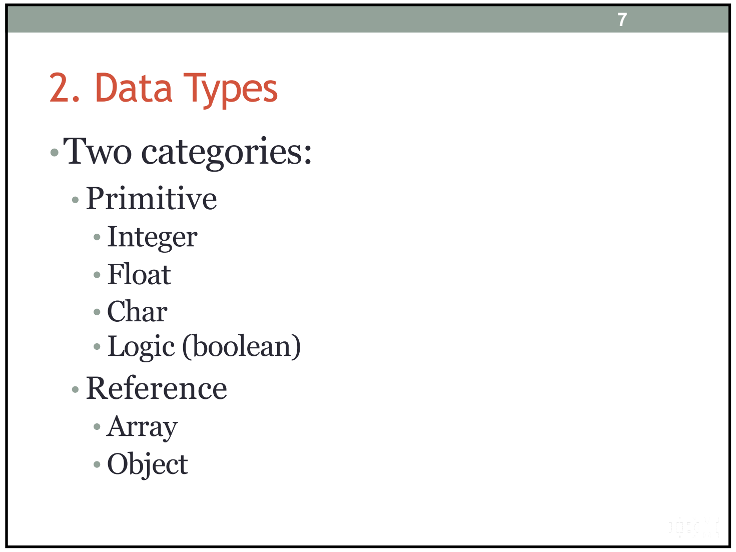
5



Content

1. Identifiers
2. Data Types
3. Operators
4. Control Statements
5. Arrays

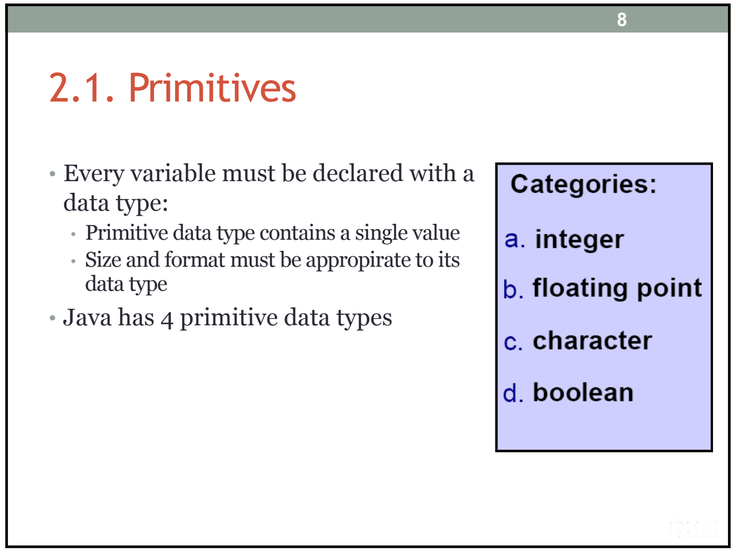
6



2. Data Types

- Two categories:
  - Primitive
    - Integer
    - Float
    - Char
    - Logic (boolean)
  - Reference
    - Array
    - Object

7



2.1. Primitives

- Every variable must be declared with a data type:
  - Primitive data type contains a single value
  - Size and format must be appropriate to its data type
- Java has 4 primitive data types

Categories:

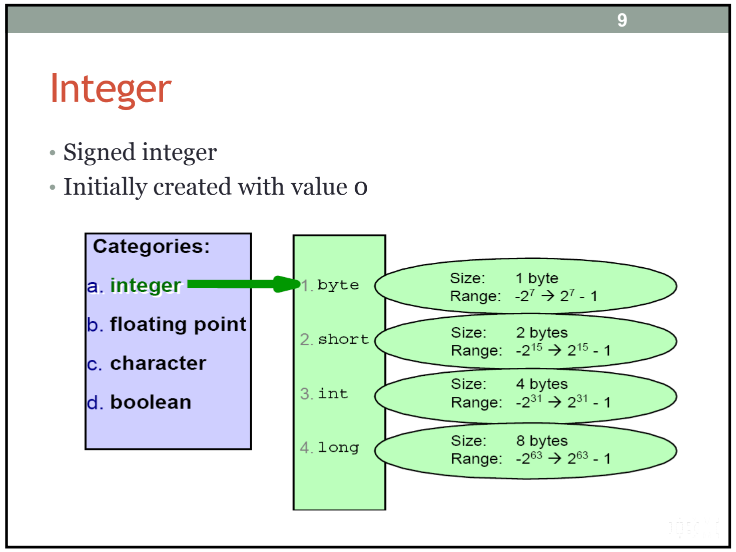
a. **integer**

b. **floating point**

c. **character**

d. **boolean**

8



9

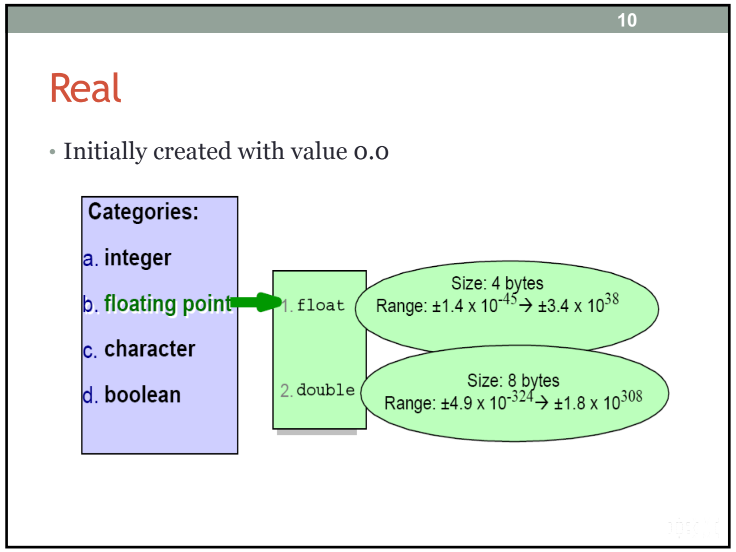
Integer

- Signed integer
- Initially created with value 0

Categories:
a. integer
b. floating point
c. character
d. boolean

1. byte — Size: 1 byte; Range: $-2^7 \rightarrow 2^7 - 1$
2. short — Size: 2 bytes; Range: $-2^{15} \rightarrow 2^{15} - 1$
3. int — Size: 4 bytes; Range: $-2^{31} \rightarrow 2^{31} - 1$
4. long — Size: 8 bytes; Range: $-2^{63} \rightarrow 2^{63} - 1$

9



10

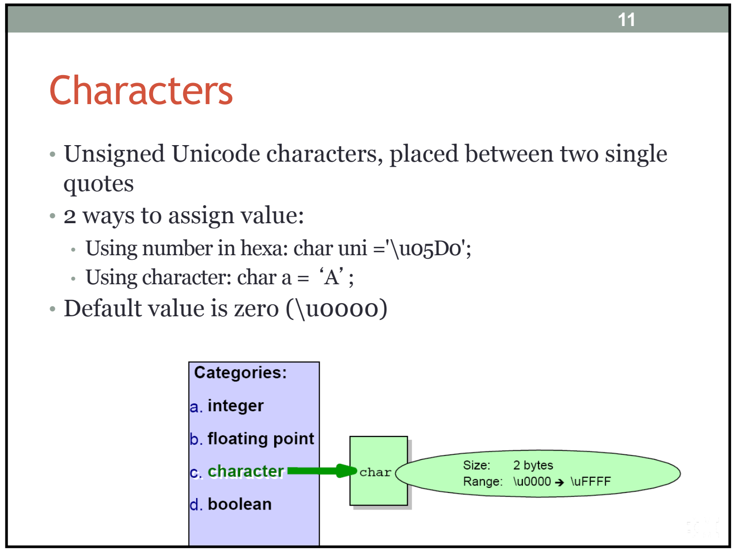
Real

- Initially created with value 0.0

Categories:
a. integer
b. floating point
c. character
d. boolean

1. float — Size: 4 bytes; Range: $\pm 1.4 \times 10^{-45} \rightarrow \pm 3.4 \times 10^{38}$
2. double — Size: 8 bytes; Range: $\pm 4.9 \times 10^{-324} \rightarrow \pm 1.8 \times 10^{308}$

10



11

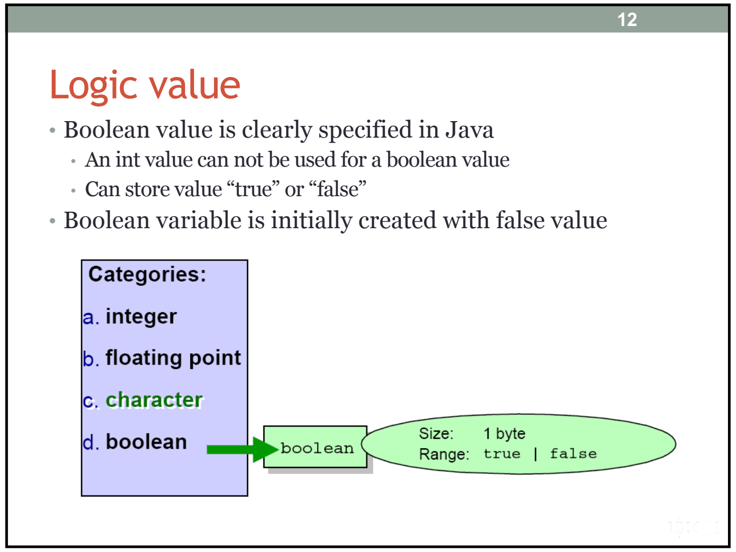
Characters

- Unsigned Unicode characters, placed between two single quotes
- 2 ways to assign value:
  - Using number in hexa: char uni ='\u05D0';
  - Using character: char a = ‘A’ ;
- Default value is zero (\u0000)

Categories:
a. integer
b. floating point
c. character
d. boolean

char — Size: 2 bytes; Range: \u0000 → \uFFFF

11



12

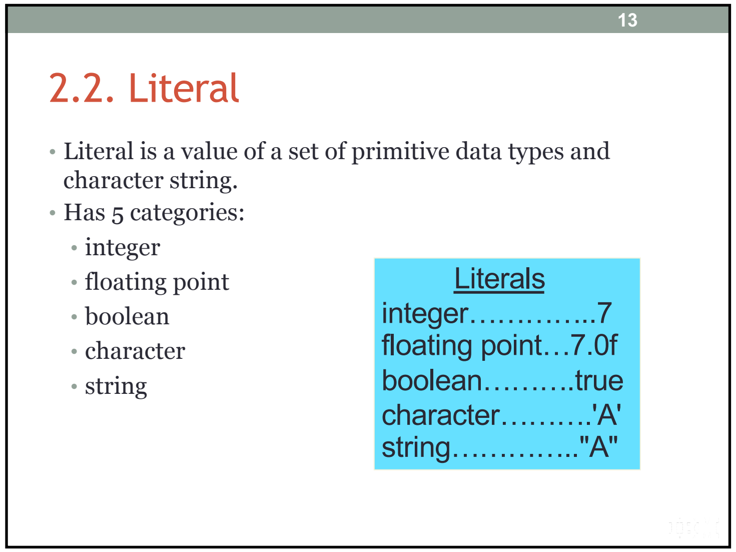
Logic value

- Boolean value is clearly specified in Java
  - An int value can not be used for a boolean value
  - Can store value “true” or “false”
- Boolean variable is initially created with false value

Categories:
a. integer
b. floating point
c. character
d. boolean

boolean — Size: 1 byte; Range: true | false

12



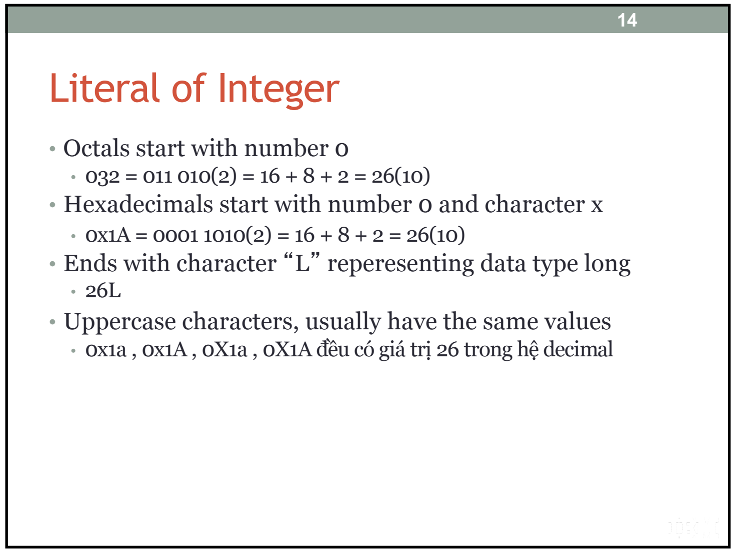
13

2.2. Literal

- Literal is a value of a set of primitive data types and character string.
- Has 5 categories:
  - integer
  - floating point
  - boolean
  - character
  - string

Literals
integer…………..7
floating point…7.0f
boolean……….true
character……….'A'
string…………..."A"

13

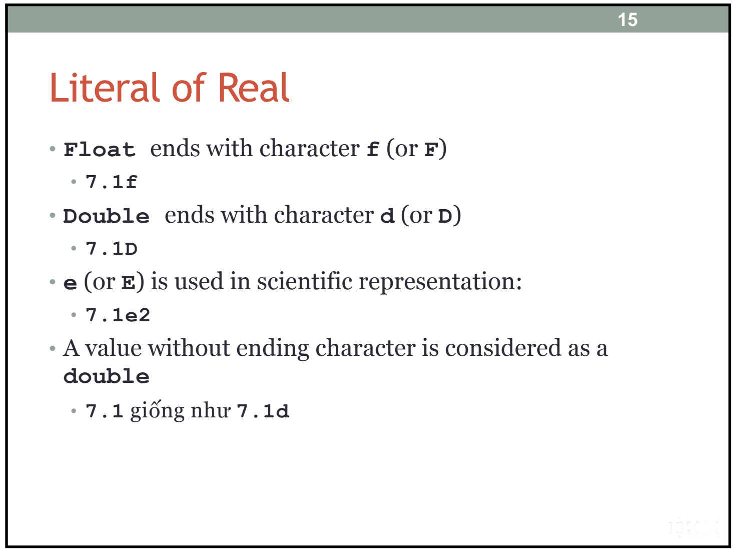


14

Literal of Integer

- Octals start with number 0
  - 032 = 011 010(2) = 16 + 8 + 2 = 26(10)
- Hexadecimals start with number 0 and character x
  - 0x1A = 0001 1010(2) = 16 + 8 + 2 = 26(10)
- Ends with character "L" reperesenting data type long
  - 26L
- Uppercase characters, usually have the same values
  - 0x1a , 0x1A , 0X1a , 0X1A đều có giá trị 26 trong hệ decimal

14

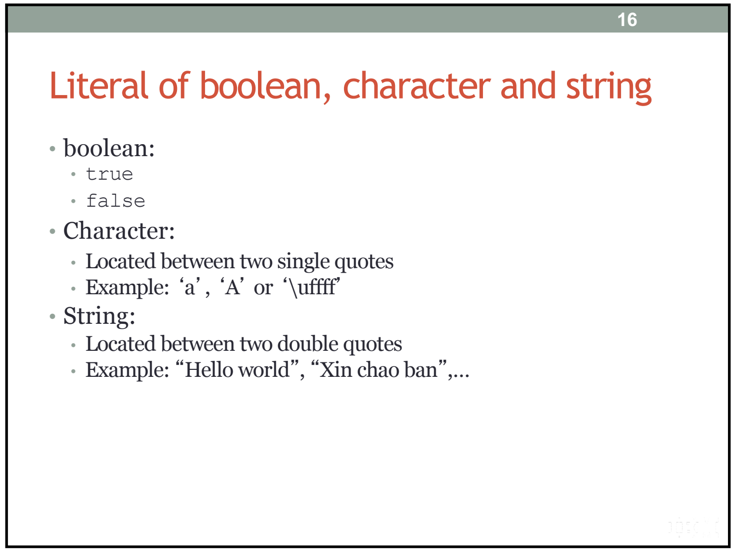


15

Literal of Real

- `Float` ends with character `f` (or `F`)
  - `7.1f`
- `Double` ends with character `d` (or `D`)
  - `7.1D`
- `e` (or `E`) is used in scientific representation:
  - `7.1e2`
- A value without ending character is considered as a `double`
  - `7.1` giống như `7.1d`

15

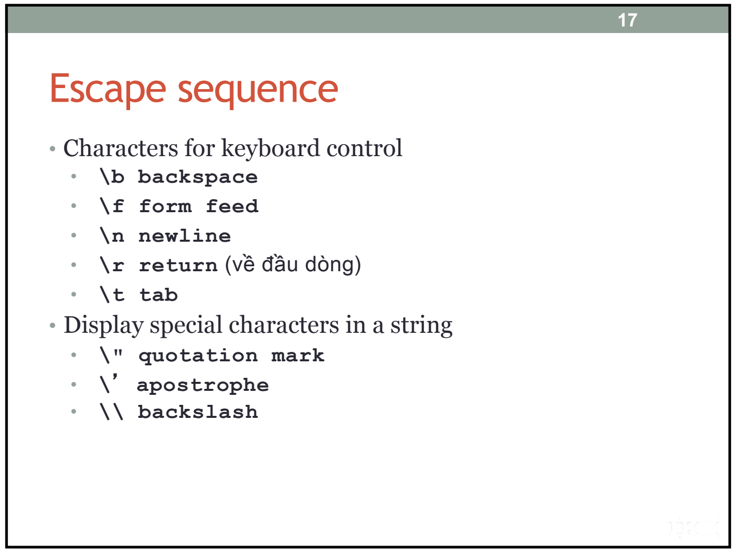


16

Literal of boolean, character and string

- boolean:
  - `true`
  - `false`
- Character:
  - Located between two single quotes
  - Example: 'a' , 'A' or '\uffff'
- String:
  - Located between two double quotes
  - Example: "Hello world", "Xin chao ban",...

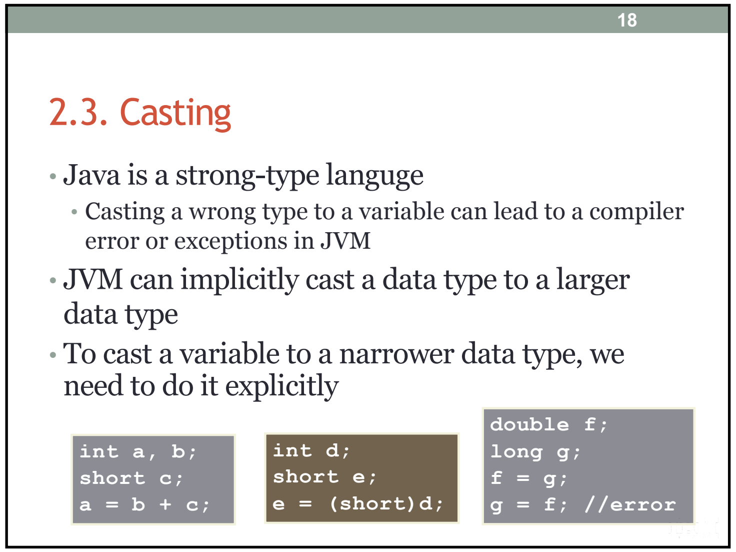
16



Escape sequence

- Characters for keyboard control
  - `\b backspace`
  - `\f form feed`
  - `\n newline`
  - `\r return` (về đầu dòng)
  - `\t tab`
- Display special characters in a string
  - `\" quotation mark`
  - `\' apostrophe`
  - `\\ backslash`

17



2.3. Casting

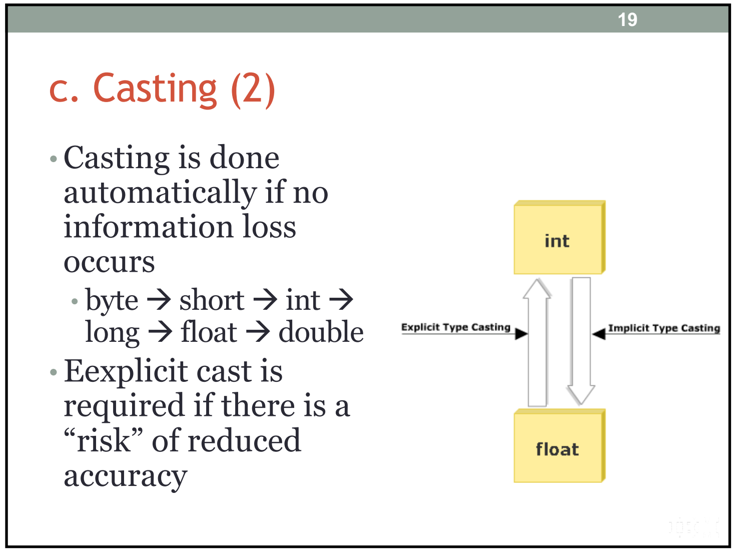
- Java is a strong-type languge
  - Casting a wrong type to a variable can lead to a compiler error or exceptions in JVM
- JVM can implicitly cast a data type to a larger data type
- To cast a variable to a narrower data type, we need to do it explicitly

```
int a, b;
short c;
a = b + c;
```

```
int d;
short e;
e = (short)d;
```

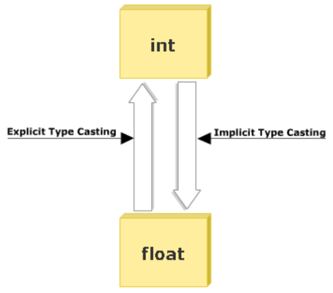
```
double f;
long g;
f = g;
g = f; //error
```

18

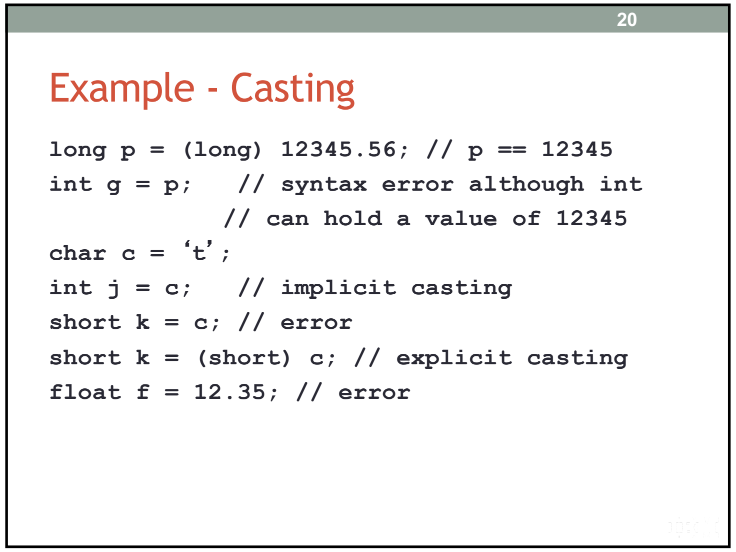


c. Casting (2)

- Casting is done automatically if no information loss occurs
  - byte → short → int → long → float → double
- Eexplicit cast is required if there is a "risk" of reduced accuracy



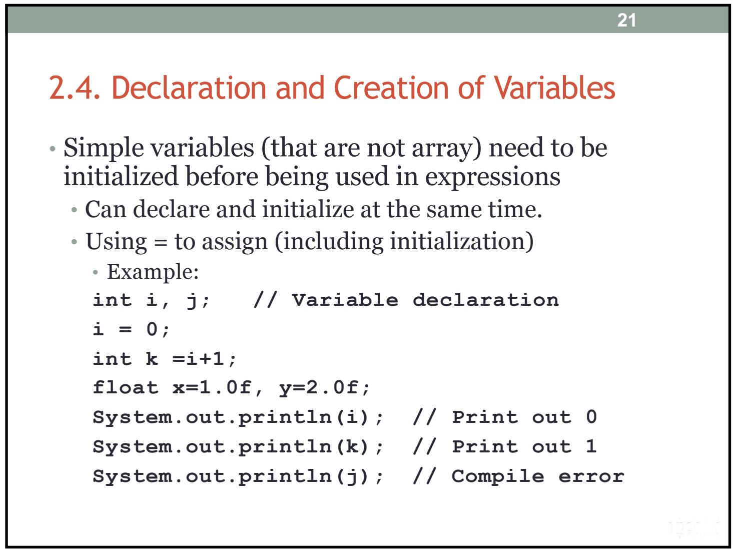
19



Example - Casting

```
long p = (long) 12345.56; // p == 12345
int g = p;    // syntax error although int
             // can hold a value of 12345
char c = 't';
int j = c;    // implicit casting
short k = c; // error
short k = (short) c; // explicit casting
float f = 12.35; // error
```

20

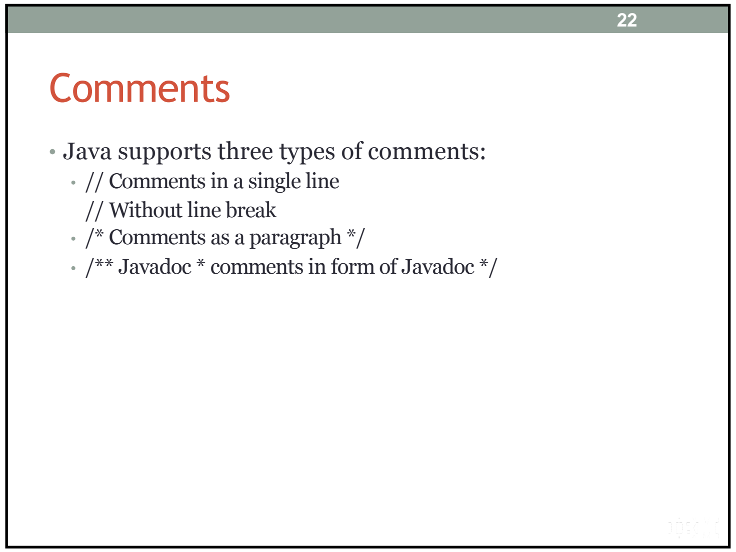


2.4. Declaration and Creation of Variables

- Simple variables (that are not array) need to be initialized before being used in expressions
  - Can declare and initialize at the same time.
  - Using = to assign (including initialization)
    - Example:

```
int i, j;   // Variable declaration
i = 0;
int k =i+1;
float x=1.0f, y=2.0f;
System.out.println(i);  // Print out 0
System.out.println(k);  // Print out 1
System.out.println(j);  // Compile error
```

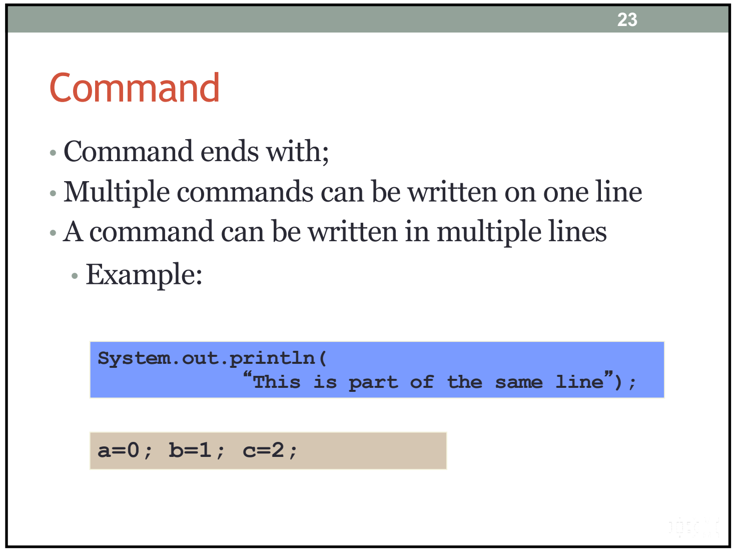
21



Comments

- Java supports three types of comments:
  - // Comments in a single line
    // Without line break
  - /* Comments as a paragraph */
  - /** Javadoc * comments in form of Javadoc */

22



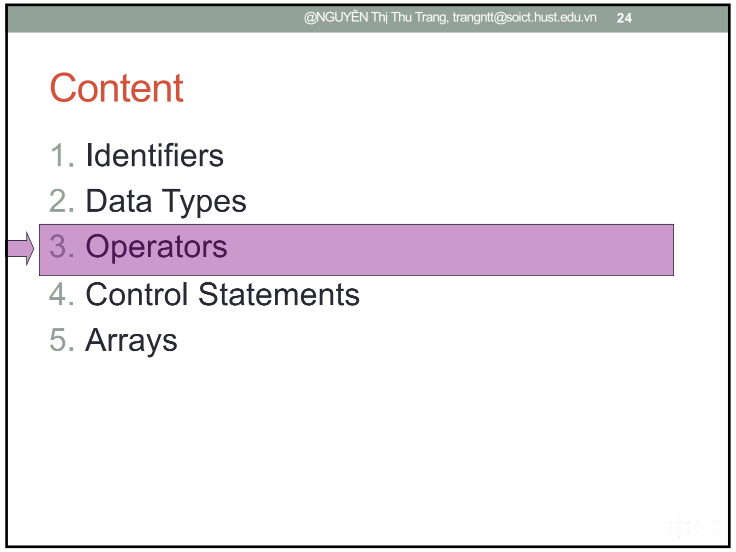
Command

- Command ends with;
- Multiple commands can be written on one line
- A command can be written in multiple lines
  - Example:

```
System.out.println(
           "This is part of the same line");
```

```
a=0; b=1; c=2;
```

23

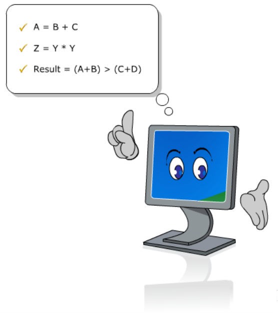


Content

24

3. Operators

- Combining single values or child expressions into a new expression, more complex and can return a value.
- Java provides the following operators:
  - Arithmetic operators
  - Bit oprator, Relational opretors
  - Logic operators
  - Assignment operators
  - Unary operators



25

3. Operators (2)

- Arithmetic operators
  - +, -, *, /, %
- Bit operators
  - AND: &, OR: |, XOR: ^, NOT: ~
  - bit: <<, >>
- Relational operators
  - ==, !=, >, <, >=, <=
- Logic operators
  - &&, ||, !

26

3. Operators (3)

- Unary operators
  - Reverse sign : +, -
  - Increase/decrease by 1 unit: ++, --
  - Negation of a logic expression: !
- Assignment operators
  - =, +=, -=, %= similar to >>, <<, &, |, ^

27

Priorities of Operators

- Define the order of performing operators – are identified by parentheses or by default as follows:
  - Postfix operators `[] .` (params) `x++ x--`
  - Unary operators `++x --x +x -x ~ !`
  - Creation or cast `new (type)x`
  - Multiplicative `* / %`
  - Additive `+ -`
  - Shift `<< >> >>> (unsigned shift)`
  - Relational `< > <= >= instanceof`
  - Equality `== !=`
  - Bitwise AND `&`
  - Bitwise exclusive OR `^`
  - Bitwise inclusive OR `|`
  - Logical AND `&&`
  - Logical OR `||`
  - Conditional (ternary) `?:`
  - Assignment `= *= /= %= += -= >>= <<= >>>= &= ^= |=`

28

Content

29

4.1. if - else statement

- Syntax

```
if (condition){
   statements;
}
else {
   statements;
}
```

- condition expression can receive boolean value
- else expression is optional

30

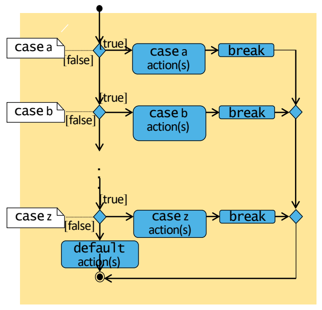
Example - Checking odd - even numbers

```
class CheckNumber
{
 public static void main(String args[])
 {
   int num =10;
   if (num %2 == 0)
            System.out.println (num+ “la so chan”);
   else
            System.out.println (num + “la so le”);
 }
}
```

31

4.2. switch - case statement

- Checking a single variable with different values and perform the corresponding case
  - break: exits switch-case command
  - Default: manages values outside the values defined in case:



32

Example - switch - case

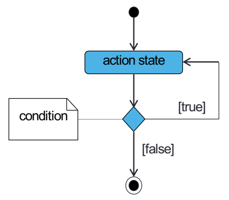
```
switch (day) {
  case 0:
  case 1:
      rule = "weekend";
      break;
  case 2:
      …
  case 6:
      rule = "weekday";
      break;
  default:
      rule = "error";
}
```

```
if (day == 0 || day == 1) {
      rule = "weekend";
} else if (day > 1 && day <7) {
      rule = "weekday";
} else {
      rule = error;
}
```

33

4.3. while and do while statements

- Perform a command or a command block while the control expression is still true
  - while() performs 0 or multiple times
  - do...while() performs at least 1 time



```
int x = 2;
while (x < 2) {
        x++;
        System.out.println(x);
}
```

```
int x = 2;
do {
        x++;
        System.out.println(x);
} while (x < 2);
```

34

Example – while loop

```
class WhileDemo{
 public static void main(String args[]){
   int a = 5,fact = 1;
   while  (a >= 1){
       fact *=a;
       a--;
    }
   System.out.println("The Factorial of 5
                            is "+fact);
 }
}
```

35

4.4. for loop

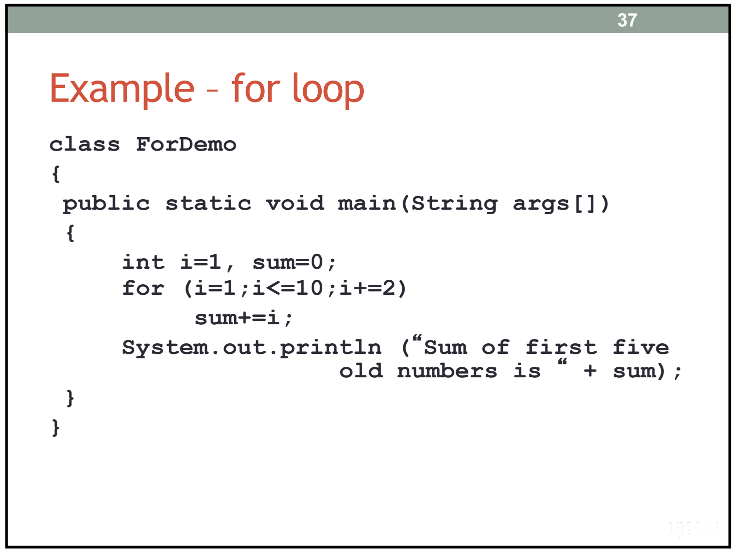
- Syntax:

```
for (start_expr; test_expr; increment_expr){
  // code to execute repeatedly
}
```

  - 3 expressions can be absent
  - A variable can be declared in for command:
    - Usually declare a counter variable
    - Usually declare in "start" expression
    - Variable scope is in the loop
- Example:

```
for (int index = 0; index < 10; index++) {
  System.out.println(index);
}
```

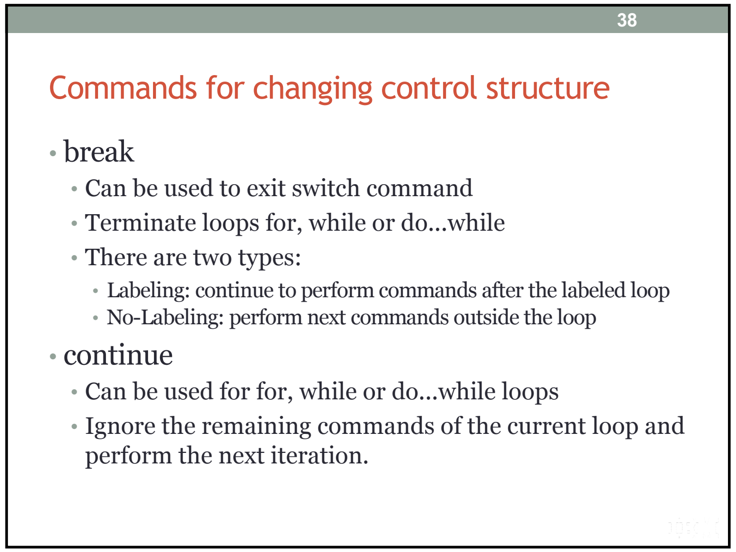
36



Example – for loop

```
class ForDemo
{
 public static void main(String args[])
 {
     int i=1, sum=0;
     for (i=1;i<=10;i+=2)
          sum+=i;
     System.out.println ("Sum of first five
                    old numbers is " + sum);
 }
}
```

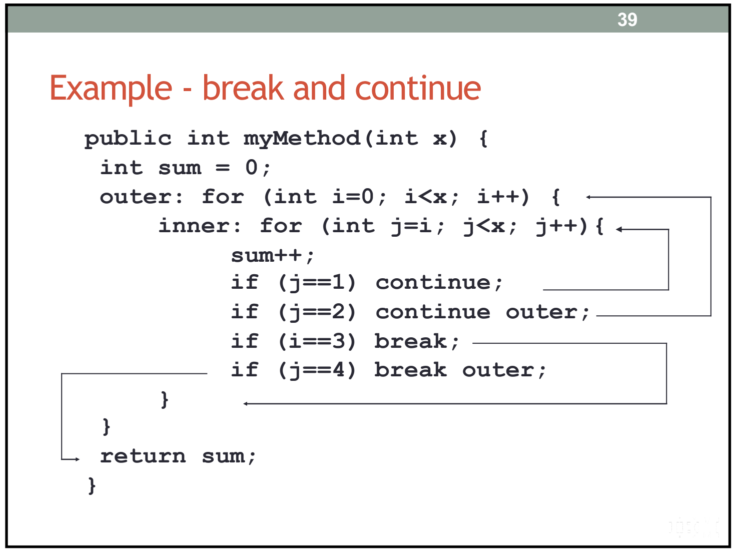
37



Commands for changing control structure

- break
  - Can be used to exit switch command
  - Terminate loops for, while or do...while
  - There are two types:
    - Labeling: continue to perform commands after the labeled loop
    - No-Labeling: perform next commands outside the loop
- continue
  - Can be used for for, while or do...while loops
  - Ignore the remaining commands of the current loop and perform the next iteration.

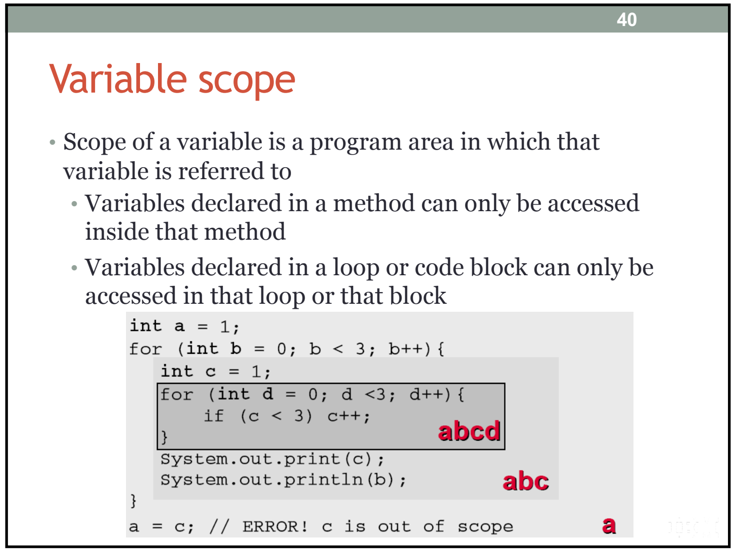
38



Example - break and continue

```
  public int myMethod(int x) {
   int sum = 0;
   outer: for (int i=0; i<x; i++) {
        inner: for (int j=i; j<x; j++){
             sum++;
             if (j==1) continue;
             if (j==2) continue outer;
             if (i==3) break;
             if (j==4) break outer;
        }
   }
   return sum;
  }
```

39



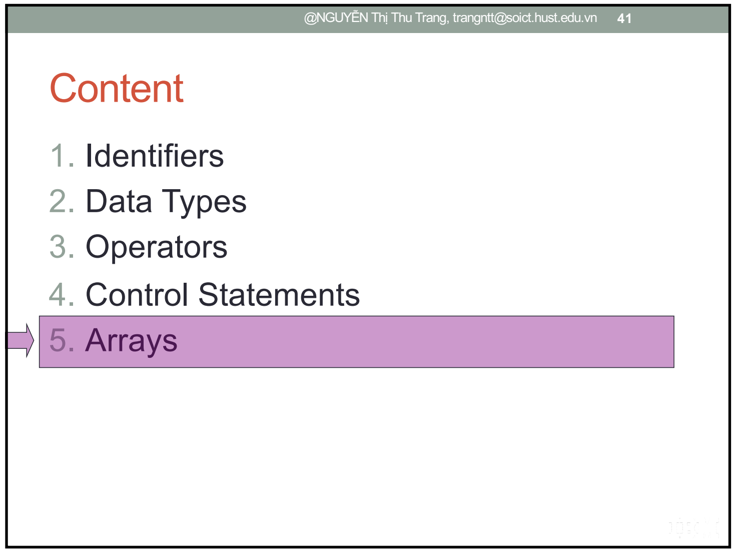
Variable scope

- Scope of a variable is a program area in which that variable is referred to
  - Variables declared in a method can only be accessed inside that method
  - Variables declared in a loop or code block can only be accessed in that loop or that block

```
int a = 1;
for (int b = 0; b < 3; b++){
    int c = 1;
    for (int d = 0; d <3; d++){
        if (c < 3) c++;
    }                                   abcd
    System.out.print(c);
    System.out.println(b);                  abc
}
a = c; // ERROR! c is out of scope              a
```

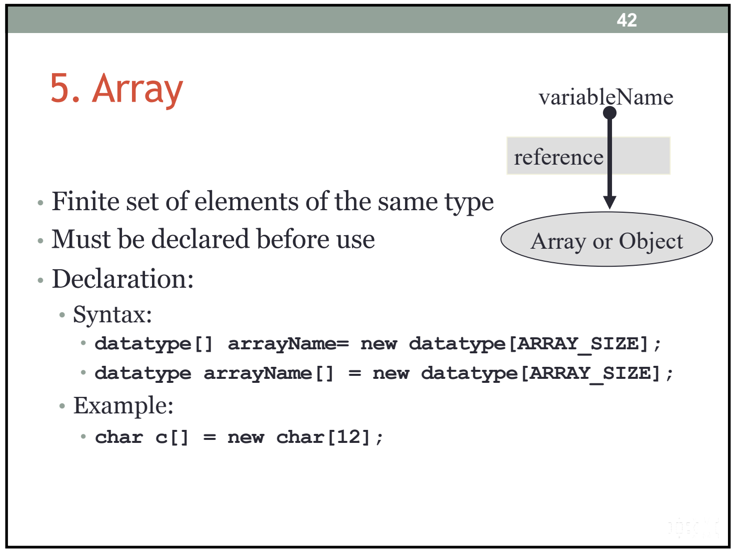
40

Page 10



Content

41

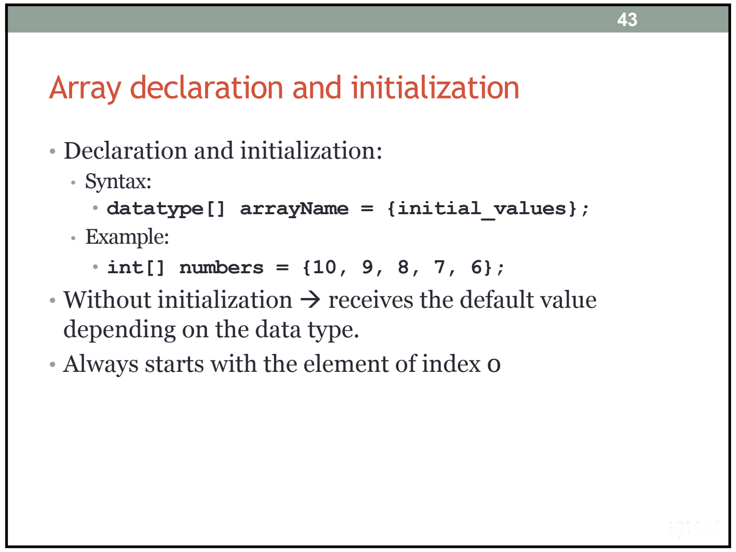


5. Array

- Finite set of elements of the same type
- Must be declared before use
- Declaration:
  - Syntax:
    - `datatype[] arrayName= new datatype[ARRAY_SIZE];`
    - `datatype arrayName[] = new datatype[ARRAY_SIZE];`
  - Example:
    - `char c[] = new char[12];`

variableName
reference
Array or Object

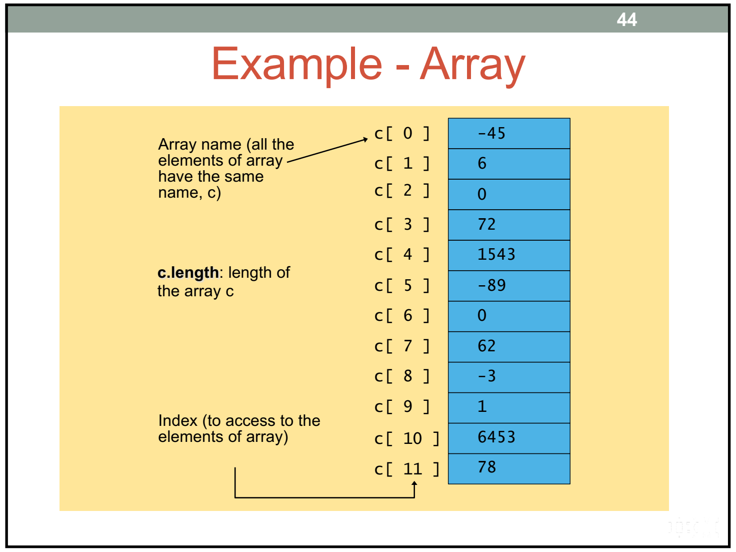
42



Array declaration and initialization

- Declaration and initialization:
  - Syntax:
    - `datatype[] arrayName = {initial_values};`
  - Example:
    - `int[] numbers = {10, 9, 8, 7, 6};`
- Without initialization → receives the default value depending on the data type.
- Always starts with the element of index 0

43

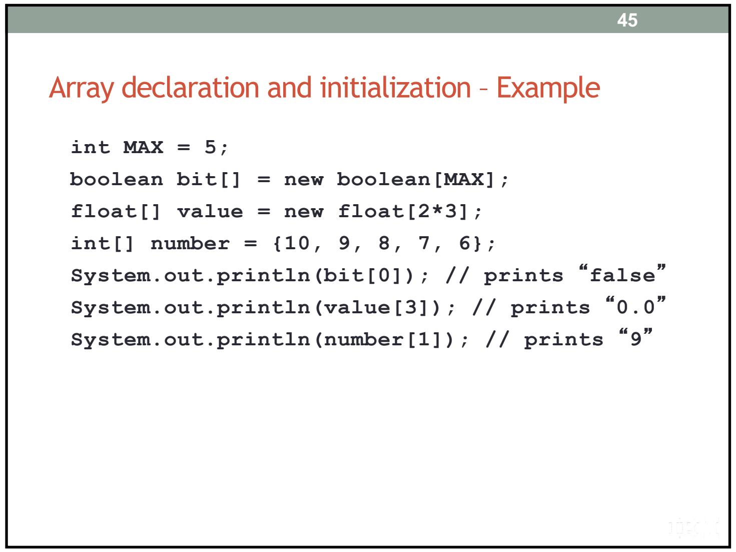


Example - Array

Array name (all the elements of array have the same name, c)
c.length: length of the array c
Index (to access to the elements of array)

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

44

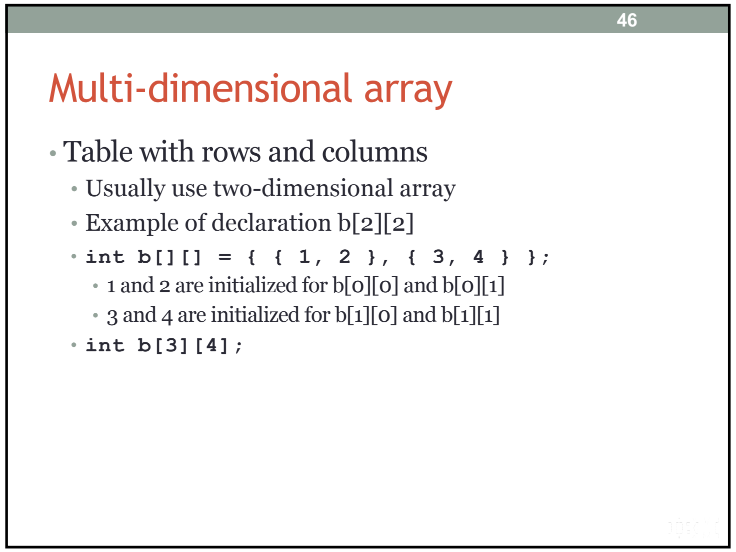


45

Array declaration and initialization - Example

```
int MAX = 5;
boolean bit[] = new boolean[MAX];
float[] value = new float[2*3];
int[] number = {10, 9, 8, 7, 6};
System.out.println(bit[0]); // prints “false”
System.out.println(value[3]); // prints “0.0”
System.out.println(number[1]); // prints “9”
```

45

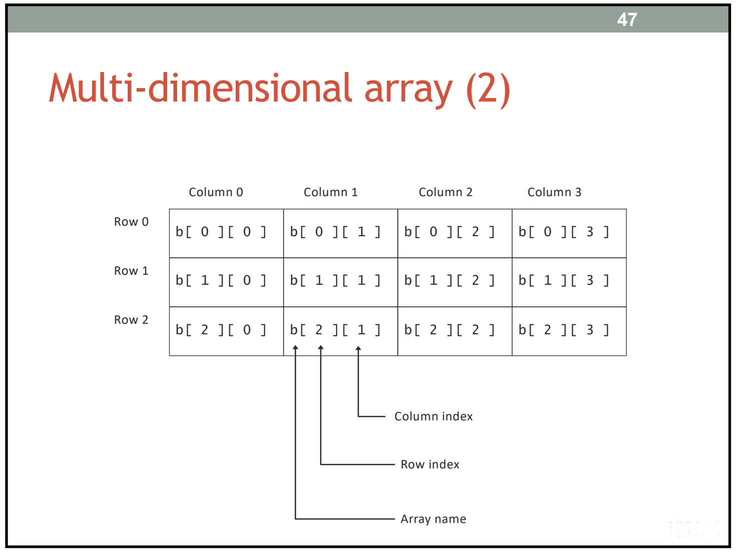


46

Multi-dimensional array

- Table with rows and columns
  - Usually use two-dimensional array
  - Example of declaration b[2][2]
  - `int b[][] = { { 1, 2 }, { 3, 4 } };`
    - 1 and 2 are initialized for b[0][0] and b[0][1]
    - 3 and 4 are initialized for b[1][0] and b[1][1]
  - `int b[3][4];`

46



47

Multi-dimensional array (2)

	Column 0	Column 1	Column 2	Column 3
Row 0	b[0][0]	b[0][1]	b[0][2]	b[0][3]
Row 1	b[1][0]	b[1][1]	b[1][2]	b[1][3]
Row 2	b[2][0]	b[2][1]	b[2][2]	b[2][3]

47