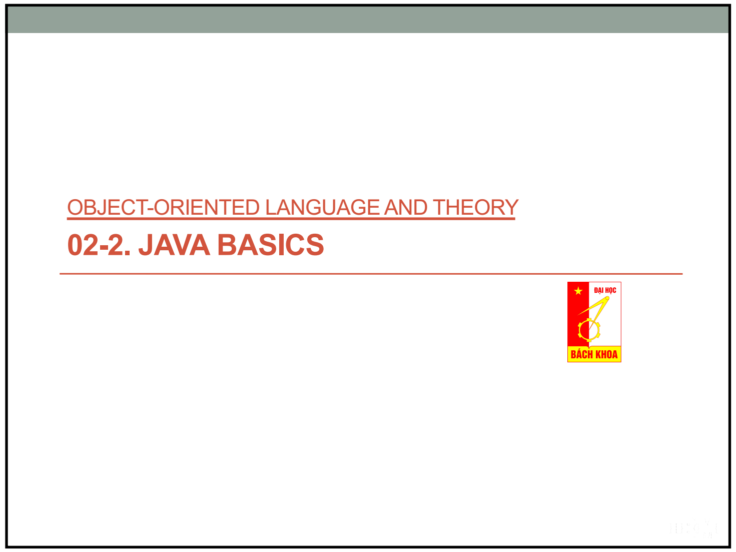
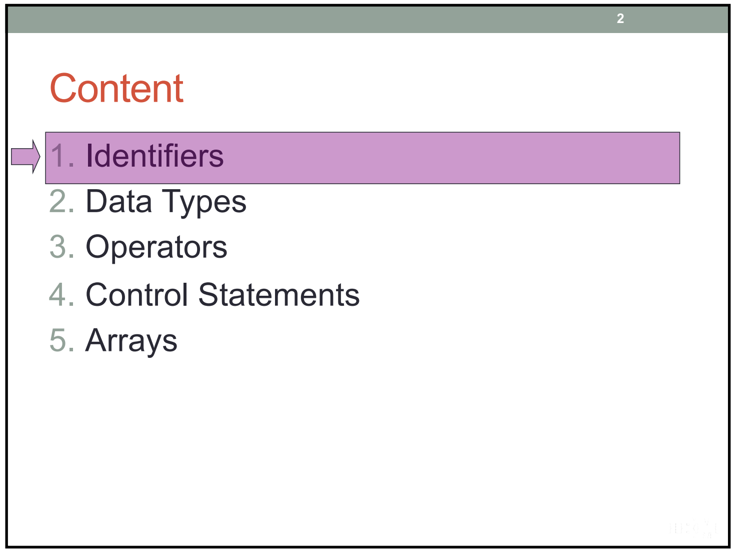

OBJECT-ORIENTED LANGUAGE AND THEORY

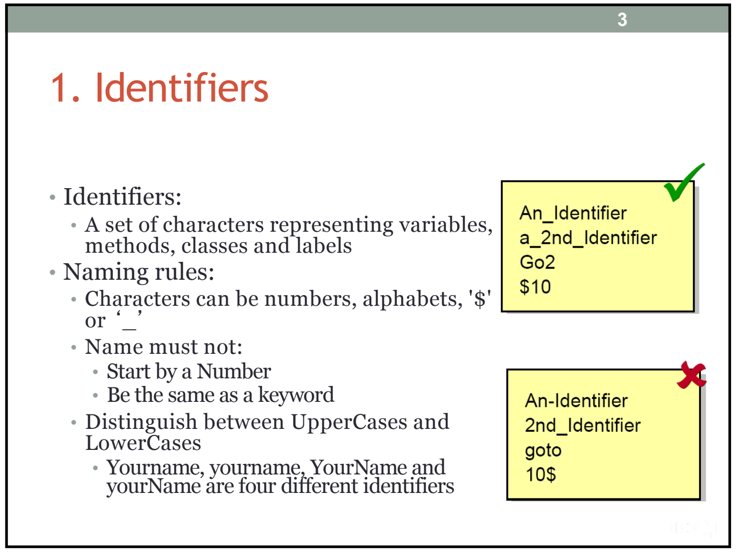
02-2. JAVA BASICS

1



Content

2



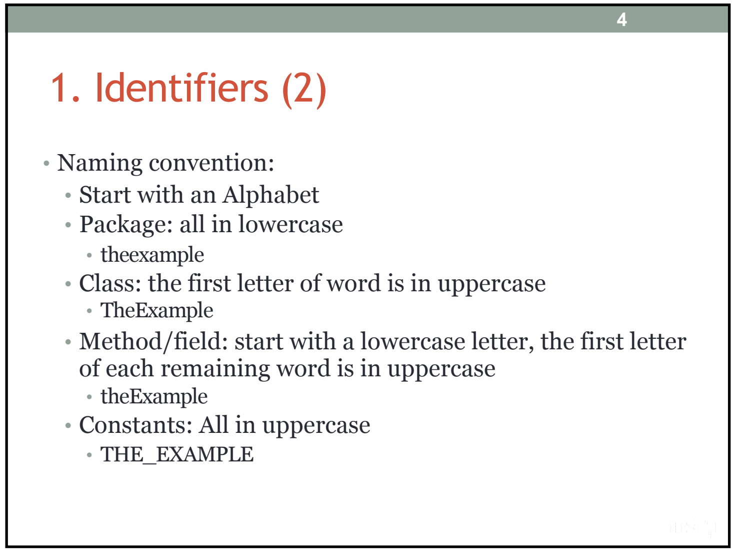
1. Identifiers

- 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 four different identifiers

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

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

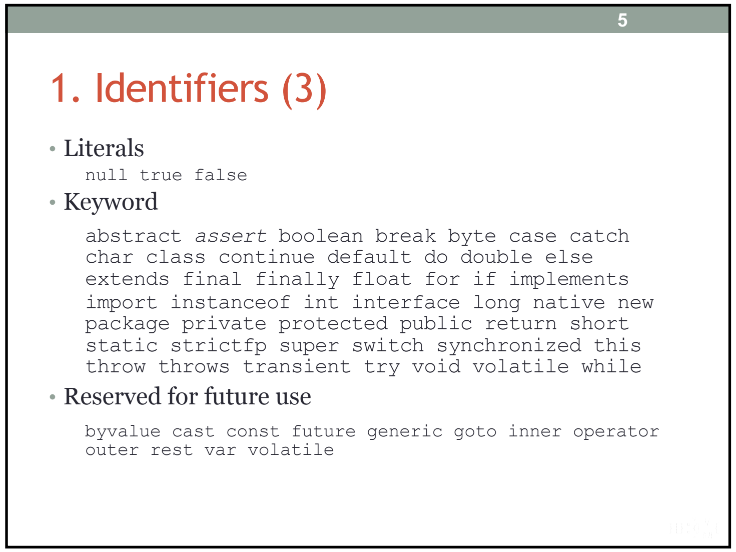
3



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 each remaining word is in uppercase
    - theExample
  - Constants: All in uppercase
    - THE_EXAMPLE

4



1. Identifiers (3)

- Literals

```
null true false
```

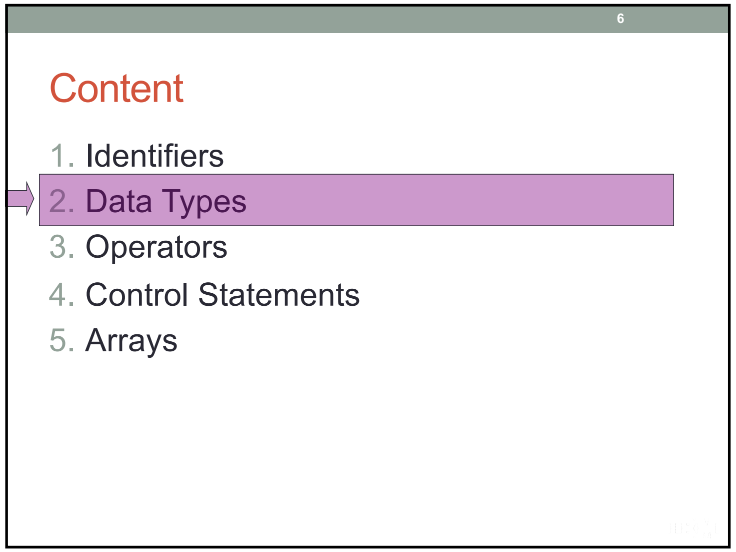
- Keyword

```
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

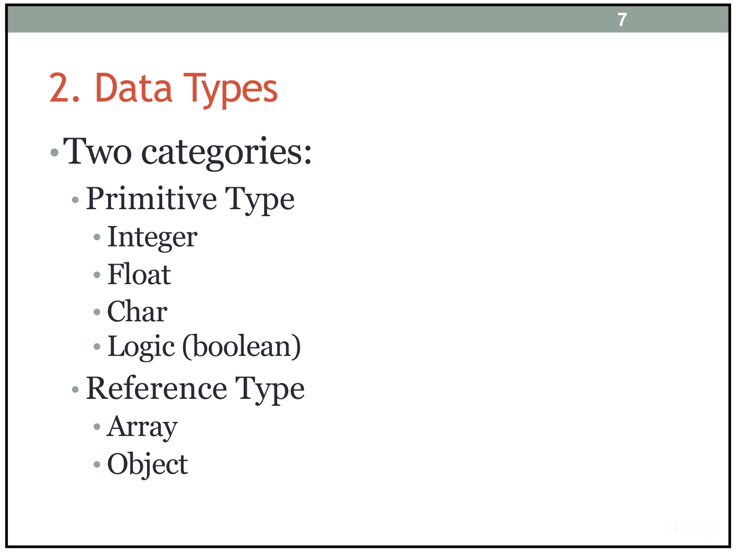
```
byvalue cast const future generic goto inner operator
outer rest var volatile
```

5



Content

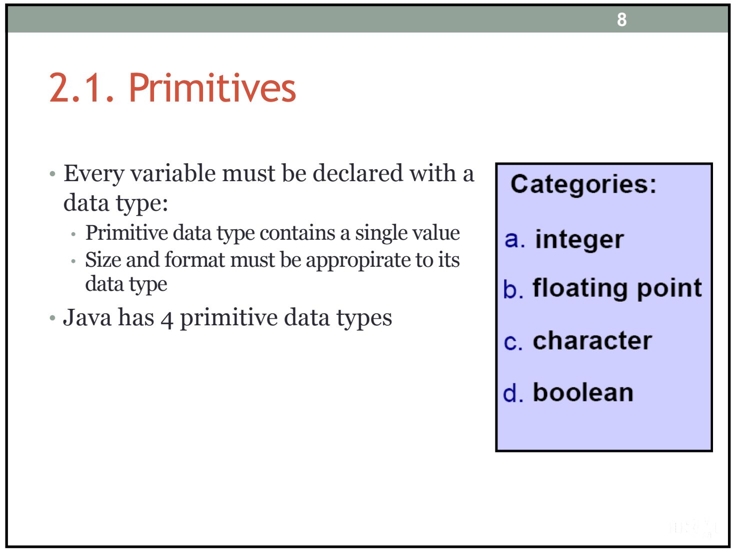
6



2. Data Types

- Two categories:
  - Primitive Type
    - Integer
    - Float
    - Char
    - Logic (boolean)
  - Reference Type
    - 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**

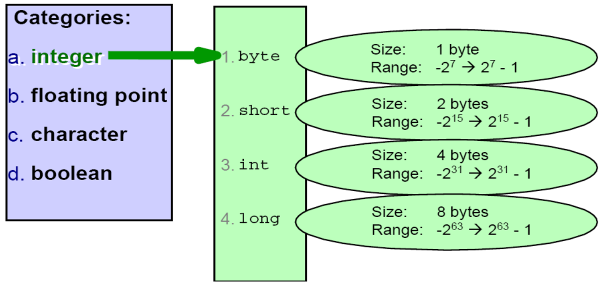
c. **character**

d. **boolean**

8

Integer

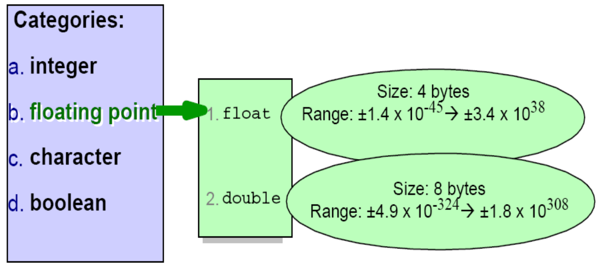
- Signed integer
- Initially created with value 0


Categories:
a. integer
b. floating point
c. character
d. boolean
1. byte
2. short
3. int
4. long
Size: 1 byte Range: $-2^7 \rightarrow 2^7 - 1$
Size: 2 bytes Range: $-2^{15} \rightarrow 2^{15} - 1$
Size: 4 bytes Range: $-2^{31} \rightarrow 2^{31} - 1$
Size: 8 bytes Range: $-2^{63} \rightarrow 2^{63} - 1$

9

Real

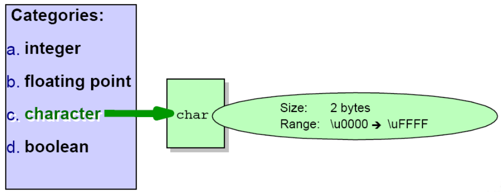
- Initially created with value 0.0


Categories:
a. integer
b. floating point
c. character
d. boolean
1. float
2. double
Size: 4 bytes Range: $\pm 1.4 \times 10^{-45} \rightarrow \pm 3.4 \times 10^{38}$
Size: 8 bytes Range: $\pm 4.9 \times 10^{-324} \rightarrow \pm 1.8 \times 10^{308}$

10

Characters

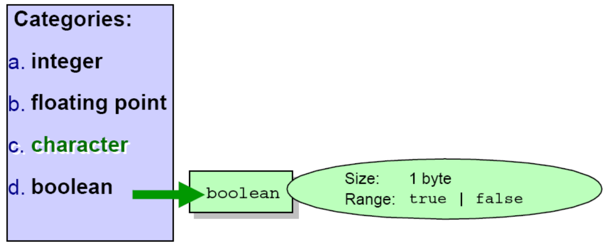
- Unsigned Unicode characters, placed between two single quotes
- 2 ways to assign value:
  - Using number in hexa: char uni ='\u05D0';
  - Using character: char a = 'A' ;
- Default value is zero (\u0000)



11

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



12

2.2. Literal

- Literal is a value of primitive types or string.
- 5 categories:
  - integer
  - floating point
  - boolean
  - character
  - string

Literals

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

13

Literal of Integer

- Octal starts with number `0`
  - `032 = 011 010(2) = 16 + 8 + 2 = 26(10)`
- Hexadecimal starts with number 0 and character x
  - `0x1A = 0001 1010(2) = 16 + 8 + 2 = 26(10)`
- Value ends with character "L" representing long data type
  - `26L`
- Case insensitive: uppercase and lowercase characters are the same
  - `0x1a , 0x1A , 0X1a , 0X1A` all are 26 in decimal

14

Literal of Real

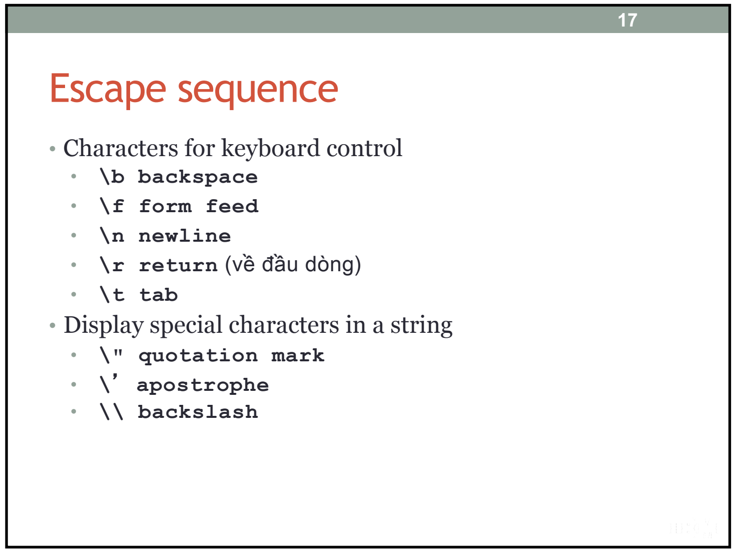
- **`Float`** ends with character **`f`** (or **`F`**)
  - **`7.1f`**
- **`Double`** ends with character **`d`** (or **`D`**)
  - **`7.1D`**
- **`e`** (or **`E`**) is used in scientific representation:
  - **`7.1e2`**
- A value without ending character is considered as a **`double`**
  - **`7.1`** is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

15

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`",…

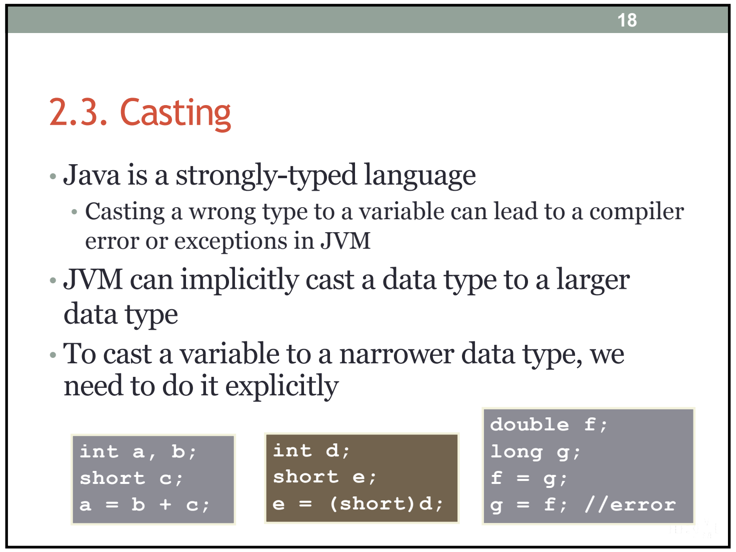
16



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

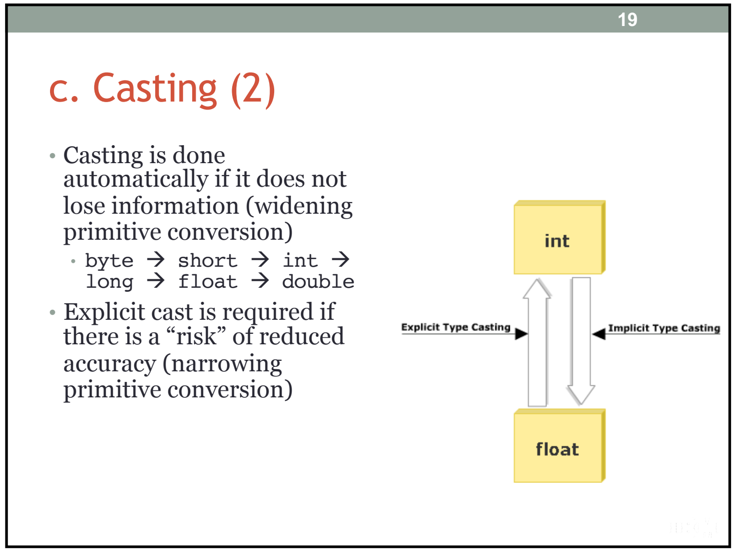
- Java is a strongly-typed language
  - Casting a wrong type to a variable can lead to a compiler error or exceptions in JVM
- JVM can implicitly cast a data type to a larger data type
- To cast a variable to a narrower data type, we need to do it explicitly

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

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

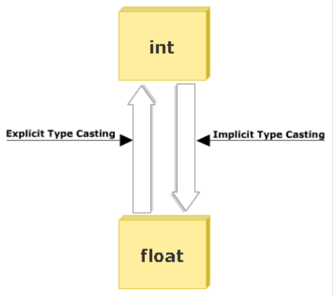
```
double f;
long g;
f = g;
g = f; //error
```

18

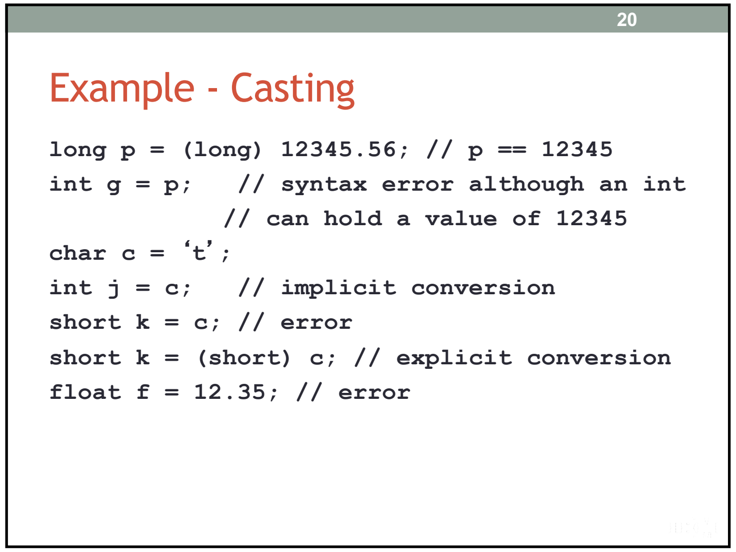


c. Casting (2)

- Casting is done automatically if it does not lose information (widening primitive conversion)
  - byte → short → int → long → float → double
- Explicit cast is required if there is a "risk" of reduced accuracy (narrowing primitive conversion)



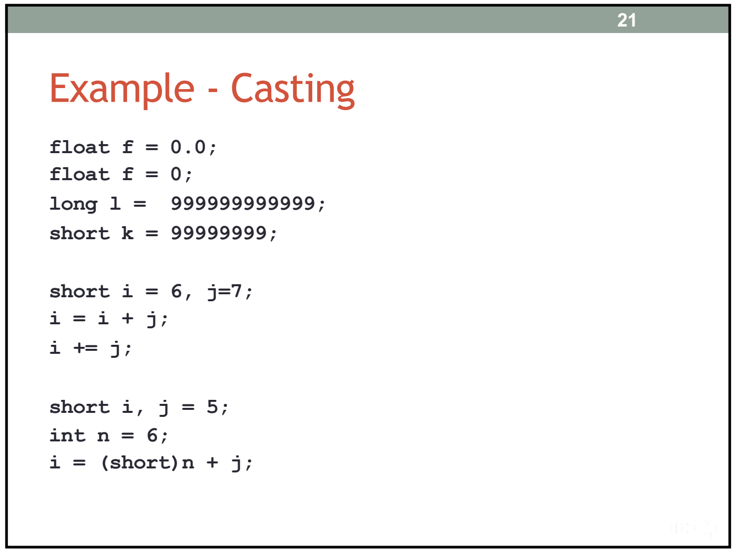
19



Example - Casting

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

20



21

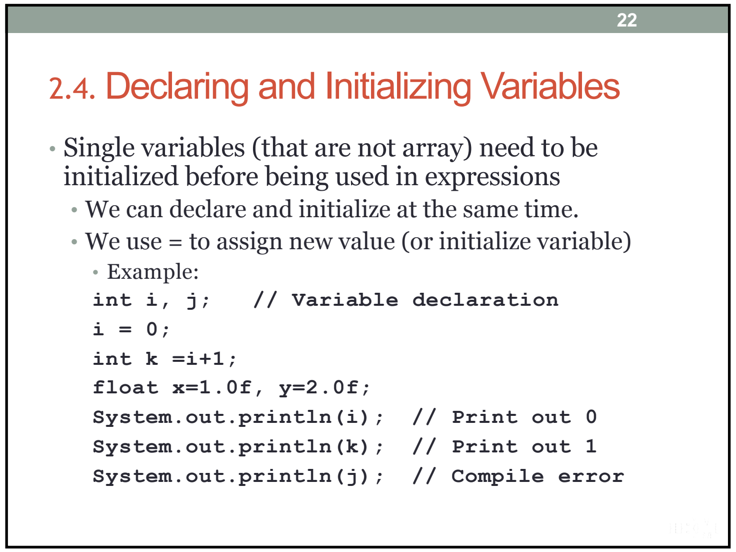
Example - Casting

```
float f = 0.0;
float f = 0;
long l =  999999999999;
short k = 99999999;

short i = 6, j=7;
i = i + j;
i += j;

short i, j = 5;
int n = 6;
i = (short)n + j;
```

21



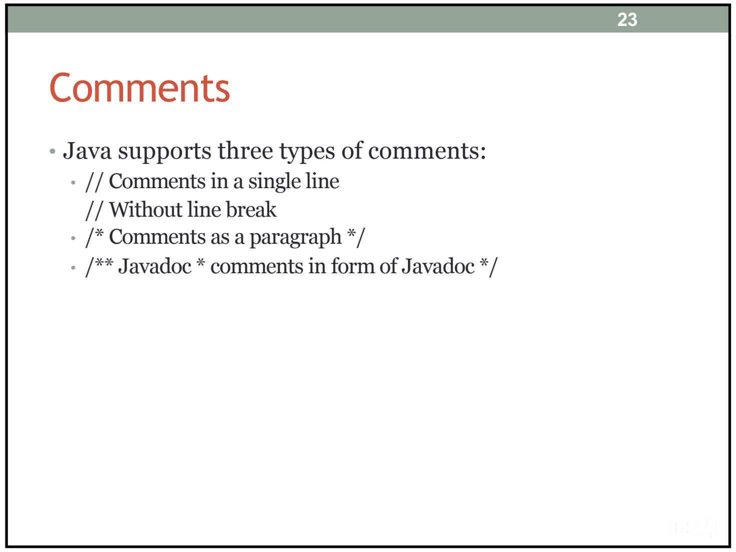
22

2.4. Declaring and Initializing Variables

- Single variables (that are not array) need to be initialized before being used in expressions
  - We can declare and initialize at the same time.
  - We use = to assign new value (or initialize variable)
    - 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
```

22

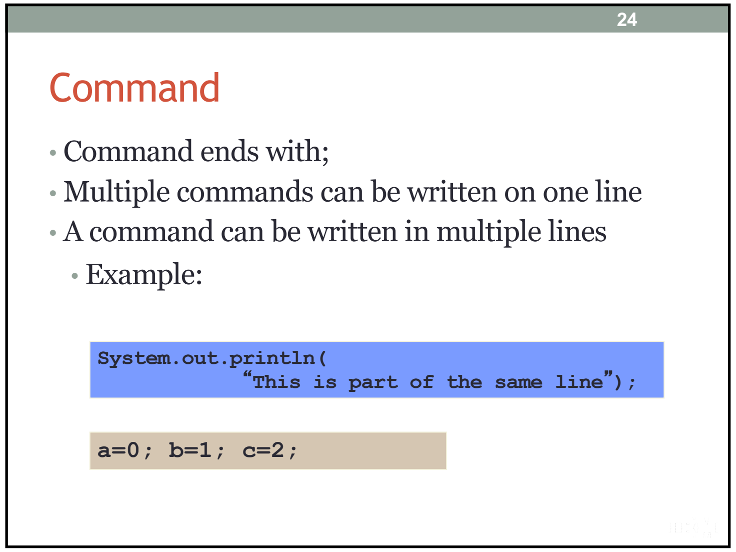


23

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 */

23



24

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");
```

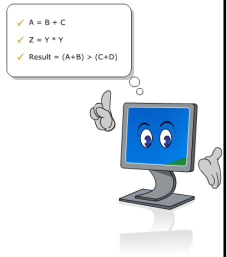
```
a=0; b=1; c=2;
```

24

Content

3. Operators

- An operator combines single values or child expressions into a more complex expression. It can return a value.
- Java provides the following operators:
  - Arithmetic operators
  - Bitwise operator, relational operators
  - Logical operators
  - Assignment operators
  - Unary operators

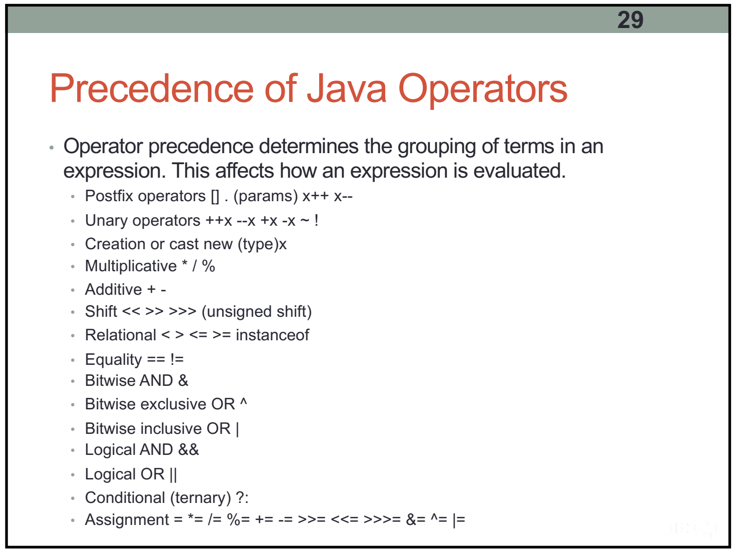


3. Operators (2)

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

3. Operators (3)

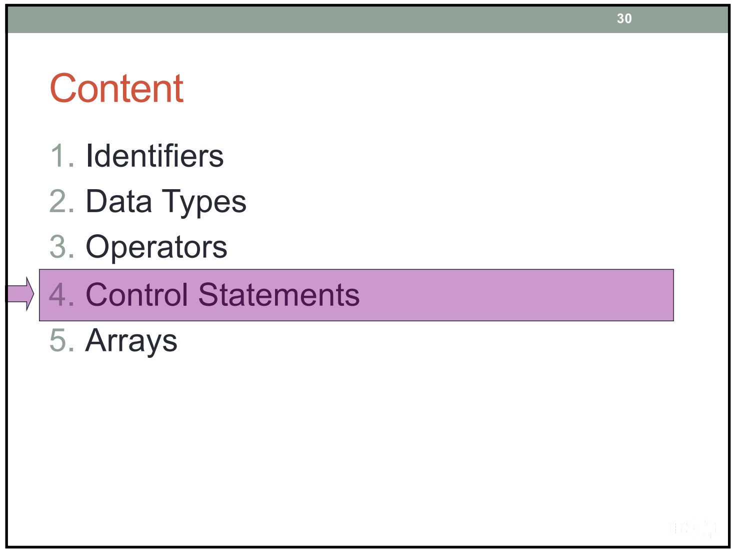
- Unary operators
  - Reverse sign : +, -
  - Increase/decrease by 1 unit: ++, --
  - Negation of a logic expression: !
- Assignment operators
  - =, +=, -=, %=
  - >>=, <<=, &=, |=, ^=



Precedence of Java Operators

- Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated.
  - 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 = *= /= %= += -= >>= <<= >>>= &= ^= |=

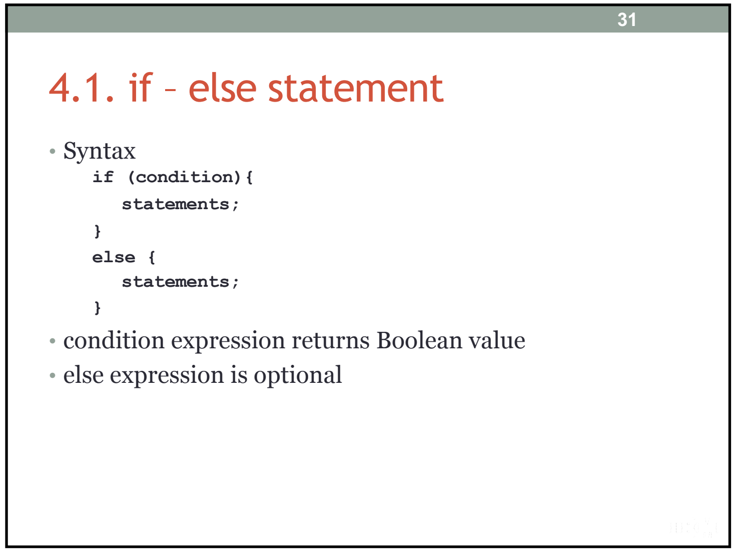
29



Content

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

30



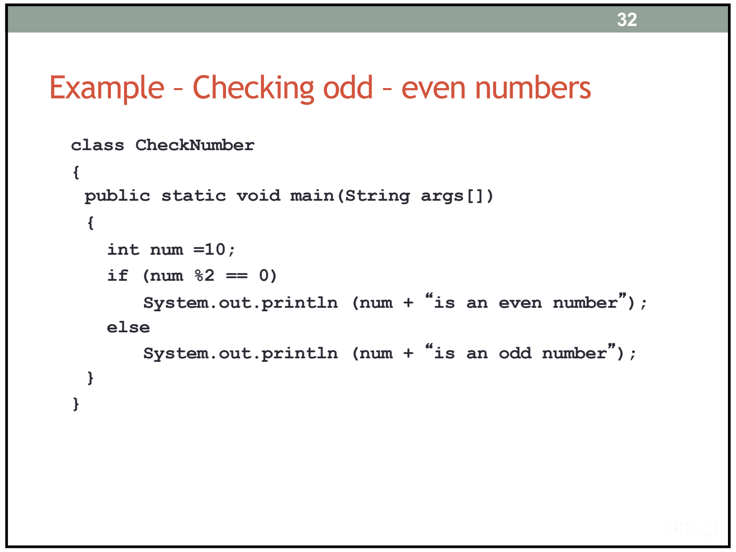
4.1. if - else statement

- Syntax

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

- condition expression returns Boolean value
- else expression is optional

31



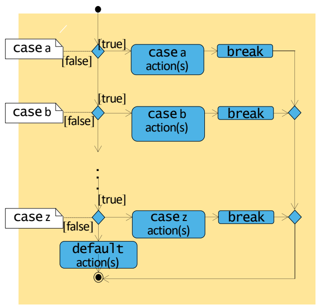
Example - Checking odd - even numbers

```
class CheckNumber
{
 public static void main(String args[])
 {
   int num =10;
   if (num %2 == 0)
      System.out.println (num + “is an even number”);
   else
      System.out.println (num + “is an odd number”);
 }
}
```

32

4.2. switch – case statement

- Check 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
- The variable used in a switch statement can only be integers, convertable integers (byte, short, char), strings and enums.



Example - switch - case

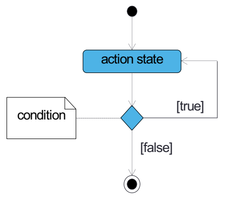
```
switch (day) {
  case 0:
  case 1:
      rule = "weekend";
      break;
  case 2:
      …
  case 6:
      rule = "weekday";
      break;
  default:
      rule = "error";
}
```

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

4.3. while and do while statements

- Perform a command or a command block as long as a given condition is 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);
```

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);
  }
}
```

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 the “start” expression
    - Variable scope is in the loop
- Example:
```
for (int index = 0; index < 10; index++) {
  System.out.println(index);
}
```

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);
 }
}
```

Loop control statements

- break
  - Can also be used to exit switch command
  - Terminate loops (**for, while** or **do...while**)
  - There are two types:
    - labeled: continue to perform commands after the labeled loop
    - unlabeled: perform next commands outside the loop containing the break statement
- continue
  - Can be used for **for, while** or **do...while** loops
  - Ignore the remaining commands of the current loop and perform the next iteration.

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;
  }
```

Variable scope

- Scope of a variable is a program area in which the 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

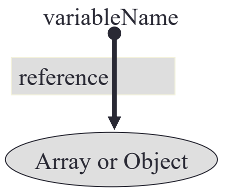
```
int a = 1;
for (int b = 0; b < 3; b++){
    int c = 1;
    for (int d = 0; d <3; d++){
        if (c < 3) c++;
    }                              abcd
    System.out.print(c);
    System.out.println(b);            abc
}
a = c; // ERROR! c is out of scope      a
```

41

Content

42

5. Array



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

43

Array declaration, instantiation, and initialization

- Declaration, instantiation 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

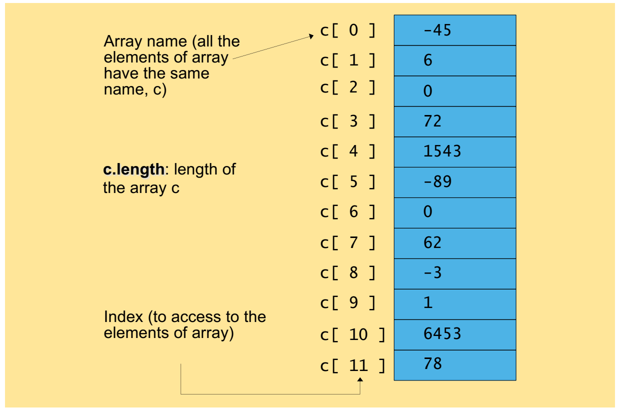
44

Example

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray {
    public static void main(String args[]) {
        int a[] = new int[5];// declaration and instantiation
        a[0] = 10;// initialization
        a[1] = 20;
        a[2] = 70;
        a[3] = 40;
        a[4] = 50;
        //traversing array
        for (int i = 0; i < a.length; i++)// length is the property of array
            System.out.println(a[i]);
    }
}
```

45

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

46

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”
```

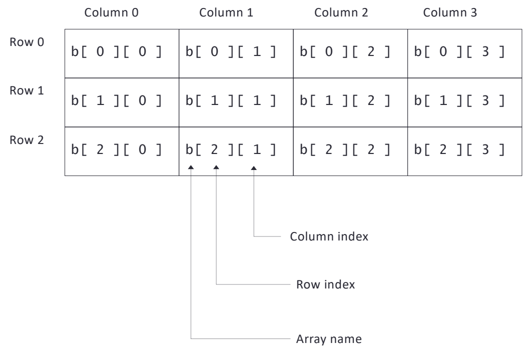
47

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];`

48

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]

49