



Java Basics – Part 1

Unit 03



Section Goals

- To learn:
 - The basic building blocks of Java
 - About variables and primitive types

Identifiers

- Identifiers are:
 - Text strings that represent variables, methods, classes or labels
 - Case-sensitive
- Characters can be digit, letter, '\$' or '_'
- Identifiers cannot:
 - Begin with a digit
 - Be the same as a reserved word.

An_Identifier ✓
a_2nd_Identifier
Go2
\$10

An-Identifier ✗
2nd_Identifier
goto
10\$

Java is Case-Sensitive

- Yourname, yourname, yourName, YourName
 - These are four different identifiers
- Conventions:
 - Package: all lower case
 - theexample
 - Class: initial upper case, composite words with upper case
 - TheExample
 - Method/field: initial lower, composite words with upper case
 - theExample
 - Constants: all upper case
 - THE_EXAMPLE

Reserved Words

- Literals

 - null true false

- Keywords

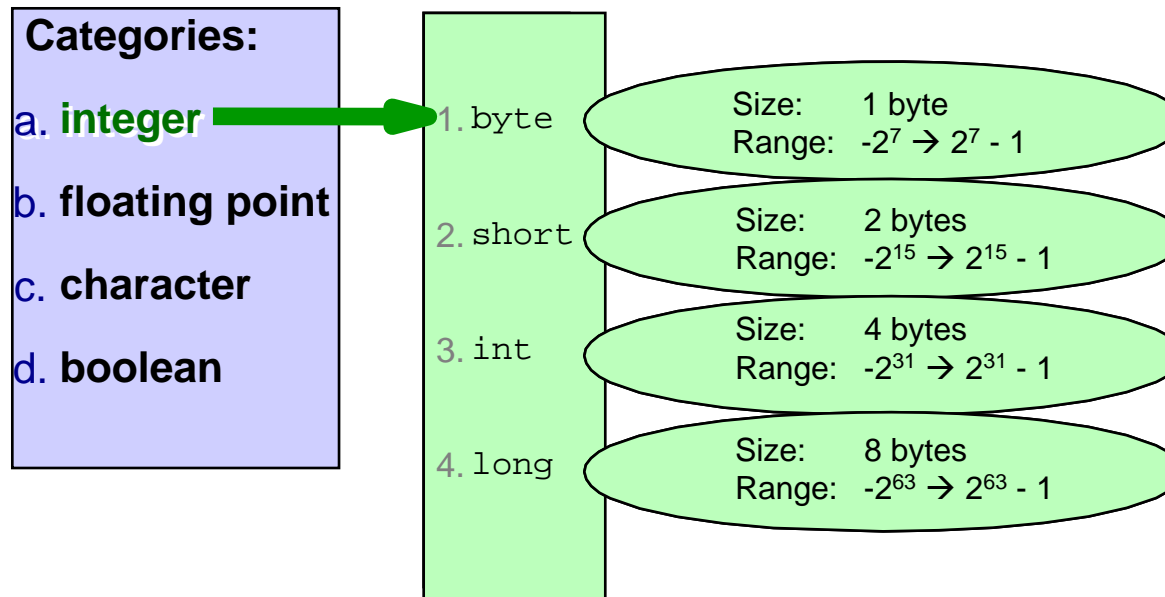
 - abstract *assert* boolean break byte case
catch char class continue default do double
else extends final finally float for if
implements import instanceof int interface
long native new package private protected
public return short static strictfp super
switch synchronized this throw throws
transient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

- Reserved for future use

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

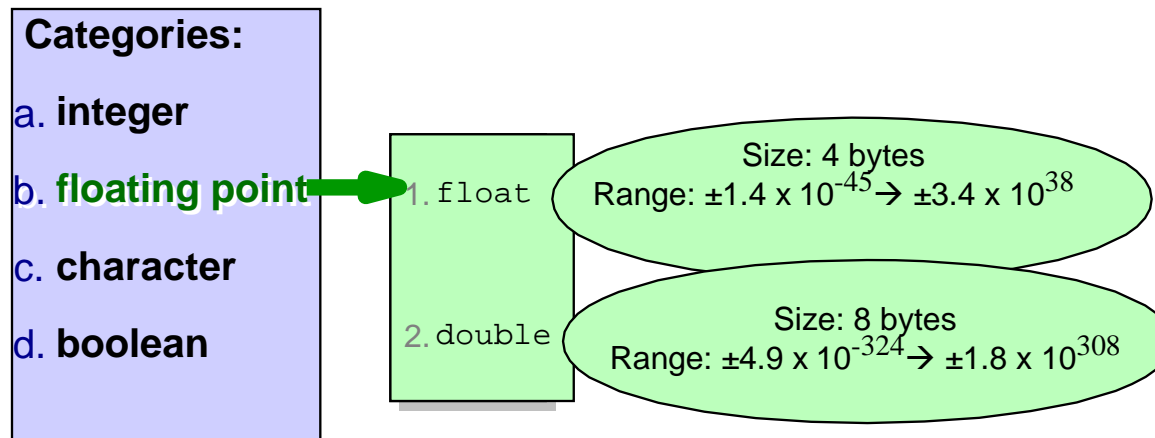
Primitives: Integers

- Signed whole numbers
- Initialized to zero



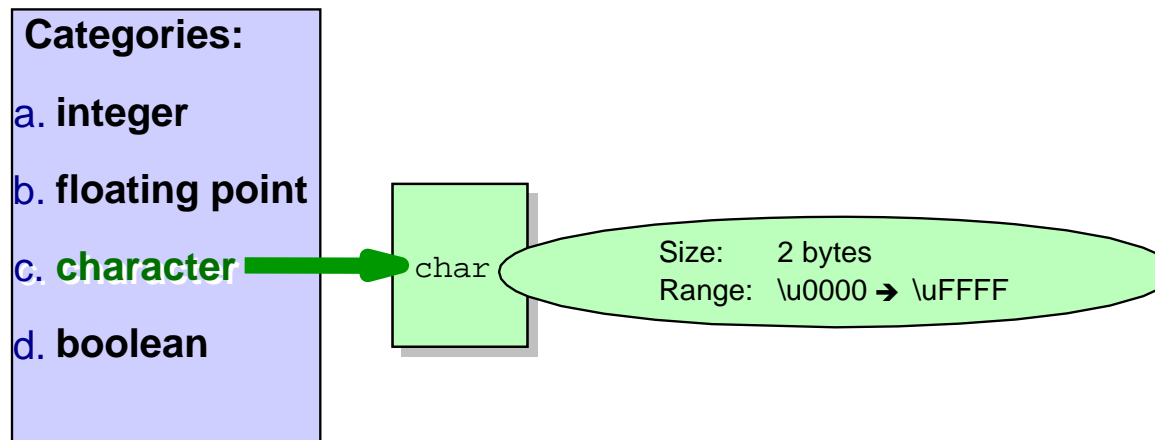
Primitives: Floating Points

- “General” numbers
 - Can have fractional parts
- Initialized to zero



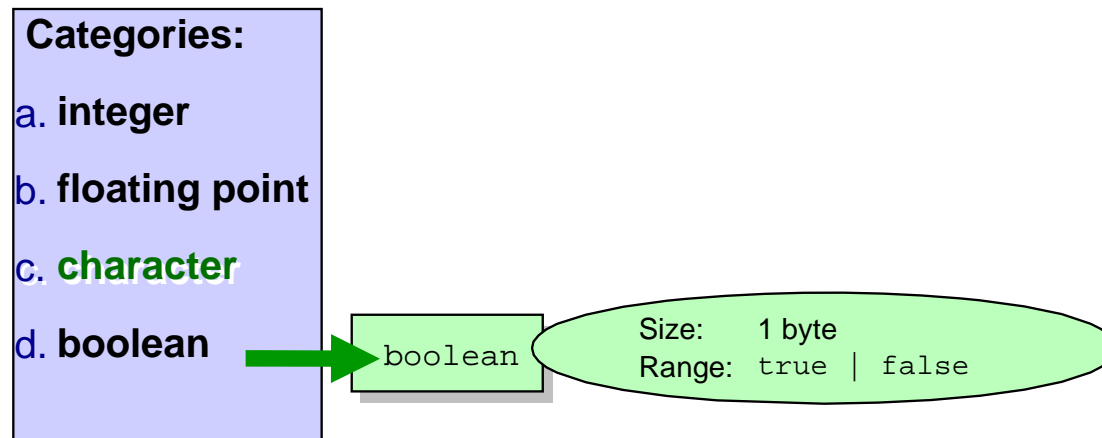
Primitives: Characters

- Char is any unsigned Unicode character
- Initialized to zero (\u0000)



Primitives: Booleans

- `boolean` values are distinct in Java
 - Can only have a `true` or `false` value
 - An `int` value can NOT be used in place of a `boolean`
- Initialized to `false`



Primitive Literals

- A literal is a value
- There are five kinds of literals:
 - Integer
 - Floating point
 - Boolean
 - Character
 - String

<u>Literals</u>	
integer.....	7
floating point...	7.0f
boolean.....	true
character.....	'A'
string.....	"A"

Primitive Literals: Integers

- Octals are prefixed with a zero:
-032
- Hexadecimals are prefixed with a zero and an x:
-0x1A
- Follow a literal with "L" to indicate a long
-26L
- **Upper and lower case are equivalent**



0x1a | 0x1A | 0X1a | 0X1A

Primitive Literals: Floating Point

- Float literals end with an f (or F):
-7.1f
- Double literals end with a d (or D):
-7.1D
- An 'e' or 'E' is used for scientific notation:
-7.1e2
- A floating point number with no final letter is a double:
-7.1 is the same as 7.1d
- Upper and lower case are equivalent

Primitive Literals: Escape Sequences

- Some keystrokes can be simulated with an escape sequence:

- \b backspace
- \f form feed
- \n newline
- \r return
- \t tab

- Some characters may need to be escaped when used in string literals

- \" quotation mark
- \' apostrophe
- \\ backslash

- Hexadecimal Unicode values can also be written '\uXXXX'

Declarations and Initialization

- Variables must be declared before they can be used
- Single value variables (not arrays) must be initialized before their first use in an expression
 - Declarations and initializations can be combined
 - Use '=' for assignment (including initialization)
- Examples:

```
int i, j;  
i = 0;  
int k=i+1;  
float x=1.0, y=2.0;  
System.out.println(i); //prints 0  
System.out.println(k); //prints 1  
System.out.println(j); //compile error
```

Arrays

- An array holds several values of the same type
- Arrays need to be declared and they
 - Have fixed size
 - The length is stated when the array is created
 - May be specified by a literal, an expression, or implicitly
 - May be optionally initialized
 - Have default values depending on their type
 - Are always zero-based, i.e., array[0] is the first element
- Examples:

```
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"
```

Operators and Precedence

- Operators are the “glue” of expressions
- Precedence – which operator is evaluated first – is determined explicitly by parentheses or implicitly as follows:
 - Postfix operators [] . (params) x++ x--
 - Unary operators ++x --x +x -x ~ !
 - Creation or cast new (type)x
 - Multiplicative * / %
 - Additive + -
 - Shift << >> >>>
 - Relational < > <= >= instanceof
 - Equality == !=
 - Bitwise AND &
 - Bitwise OR ^
 - Bitwise inclusive OR |
 - Logical AND &&
 - Logical OR ||
 - Conditional ?:
 - Assignment = *= /= %= += -=
 >>= <<= >>>= &= ^= |=

Comments

- Three kinds of comments:

```
1. // The rest of the line is a comment
   // No line breaks.

2. /* Everything between
   is a comment */

3. /** Everything between
   * is a javadoc comment */
```

- Comments do not effect the output of the program

Statements

- Terminated by a semicolon
- Several statements can be written on one line, or
- Can be split over several lines

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

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

Casting Primitives in Java –

creates a new value allowing the value to be treated as a different type (page 216 Deitel)

PRIMITIVE TYPES

Boolean

boolean 00

Character

char 0000

Integer

byte 00

short 0000

int 00000000

long 0000000000000000

Floating point

float 00000000

double 0000000000000000

(00 represents a byte)

Valid promotions

→ Cannot promote to any other primitive type

→ int, long, float, or double

→ short, int, long, float, or double (not char)

→ int, long, float, or double (not char)

→ long, float, or double

→ float or double

→ double

→ Cannot promote to any other primitive type

eclipse

Casting Primitive Types

- Casting creates a new value and allows it to be treated as a different type than its source
- Java is a strictly typed language
 - Assigning the wrong type of value to a variable could result in a compile error or a JVM exception
- The JVM can implicitly promote from a narrower type to a wider type
- To change to a narrower type, you must cast 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
```

Implicit vs. Explicit Casting

- Implicit casting is automatic when no loss of information is possible.
 - byte → short → int → long → float → double
- An explicit cast required when there is a "potential" loss of accuracy:

```
long p = (long) 12345.56;    // p == 12345
int g = p;    // illegal even though an int
               // can hold 12345
char c = 't';
int j = c;    // automatic promotion
short k = c;  // why is this an error?
short k = (short) c;    // explicit cast
float f = 12.35;    // what's wrong with this?
```

Homework 3

- **Due 9/16 11:59PM**
- **Did you do it checklist**
 - **Q1 answered**
 - **Q3 answered**
 - **Q5 answered**
 - **Q7 answered**
 - **Q9 source**
 - **Q2 answered**
 - **Q4 answered**
 - **Q6 answered**
 - **Q8 answered**
 - **Q10 source, class file, screen print of execution**
- Notes from the reading through page 65
 - 43-45 shows 3 different ways to print to the console
 - 45 shows escape sequences
 - 47 has the addition class as an example in which variable declaration is discussed, the Scanner class is introduced and primitive types are introduced.
 - 48-53 discusses various operators (assignment, binary, arithmetic, equality, relational)
 - 53 discusses precedence
 - 58 gives an short example of the if statement
- 1. Pretend you are making a program that models a card game like poker. List at least 4 objects that would be useful in such a program. Suggest methods for them to compute. There is an example UML diagram on page 87, use this format.
- 2. Write out a line of code that calls a method, `getSomething()`, on an Object, `obj`, and stores the value returned in an int, `i`.
- 3. In eclipse, how you can automatically write the get and set methods for any variable in a class. Find the place in eclipse to automatically create getters and setters and describe it here.
- 4. page 65 do self review exercise 2.3
- 5. What four characteristics does an object usually possess?
- 6. How are classes and objects related?
- 7. What does an object message correspond to in non-OO programming terms?
- 8. Define instantiation:
- 9. page 68 do problem 2.8
- 10. page 69 do problem 2.17

What You Have Learned

- In this unit, you have learned about the basic building blocks of Java syntax, including:
 - Identifiers
 - Keywords
 - Primitive data types
 - Casting primitives
 - Literals
 - Initialization
 - Operators
 - Variables
 - Arrays
 - Comments
 - Statements