Unit objectives

- After completing this unit, you should be able to:
  - Outline naming conventions used by Java programs
  - Construct a valid identifier
  - Describe the Java primitive data types, and explain how and why each one is used
  - Declare and initialize Java variables and arrays
  - Identify reserved words

Identifiers

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

✓

An_Identifier
a_2nd_Identifier
Go2
$10

✗

An-Identifier
2nd_Identifier
goto
10$

Identifiers

An *identifier* is an unlimited-length sequence of *Java letters* and *Java digits*, the first of which must be a *Java letter*.

Identifier:
  IdentifierChars but not a *Keyword* or *BooleanLiteral* or *NullLiteral*

IdentifierChars:
  JavaLetter {JavaLetterOrDigit}

JavaLetter:
  any Unicode character that is a "Java letter"

JavaLetterOrDigit:
  any Unicode character that is a "Java letter-or-digit"

A "Java letter" is a character for which the method `Character.isJavaIdentifierStart(int)` returns true.

Java is case-sensitive

- Java is case-sensitive
  - yourname, yourName, Yourname, YourName 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

A literal is the source code representation of a value of a primitive type, the String type,or the null type.

- Keywords

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

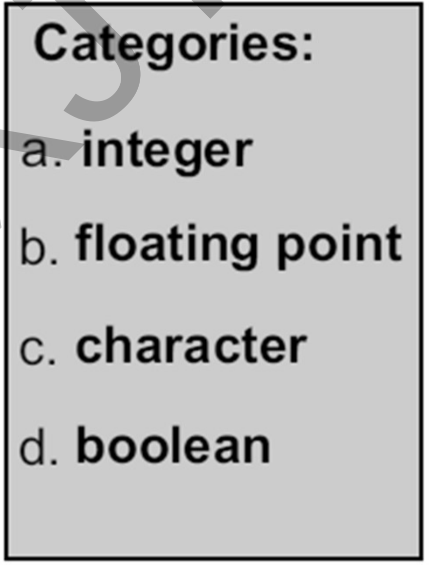
- Reserved for future use
  - const goto

Java Types

- There are two kinds of types
  - PrimitiveType
    - Integer
    - Float
    - Character
    - Boolean
  - ReferenceType
    - ClassOrInterfaceType
    - TypeVariable
    - ArrayType

Java primitives

- Every variable must have a data type
  - Primitive data types contain a single value
  - The size and format of a primitive data type are suited to its type
- Java has four categories of primitives



Categories:

a. **integer**

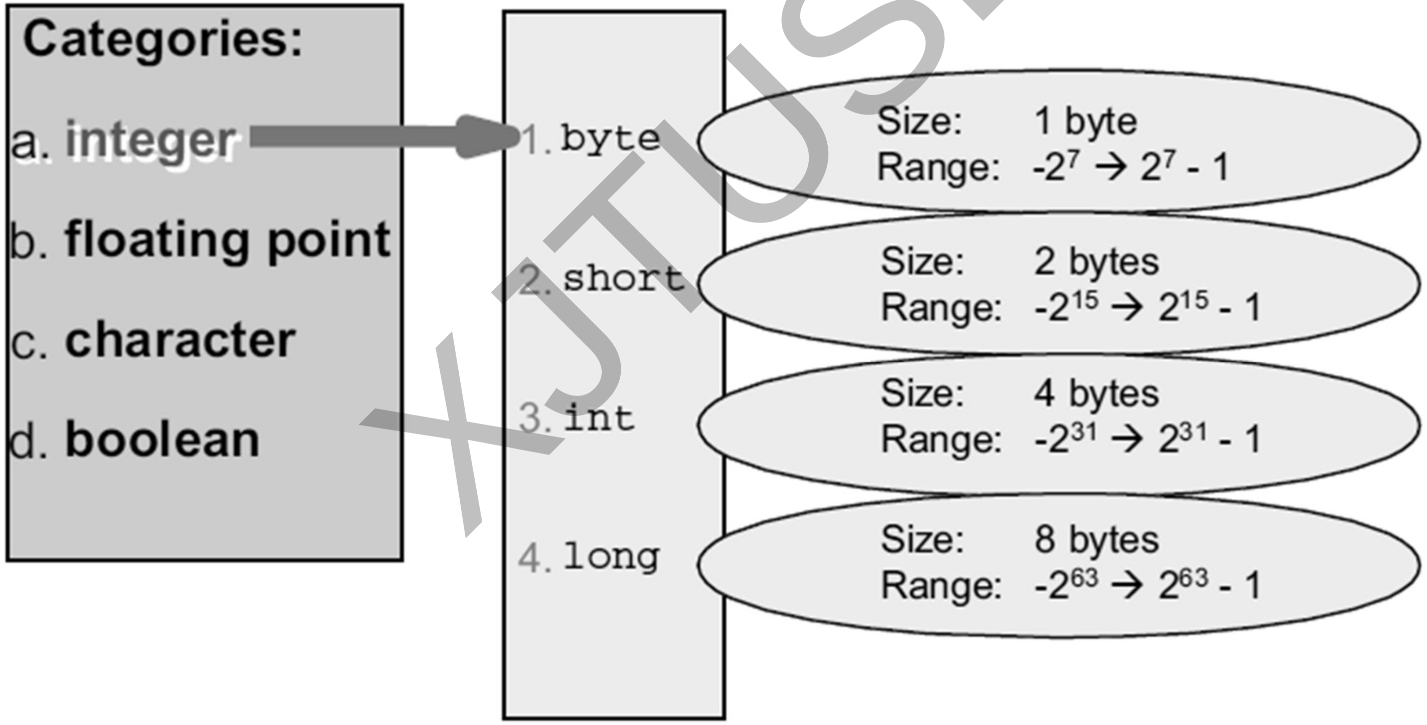
b. **floating point**

c. **character**

d. **boolean**

Primitives:integers

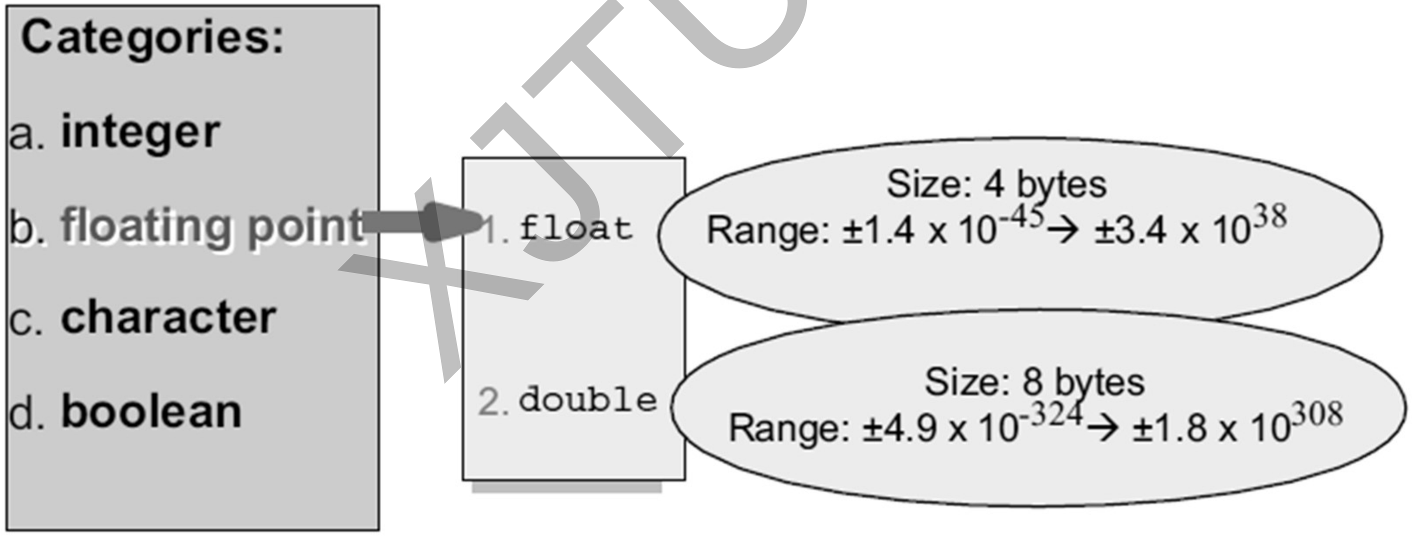
- Signed whole numbers
- Initialized to zero



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: $-2^{63} \rightarrow 2^{63} - 1$

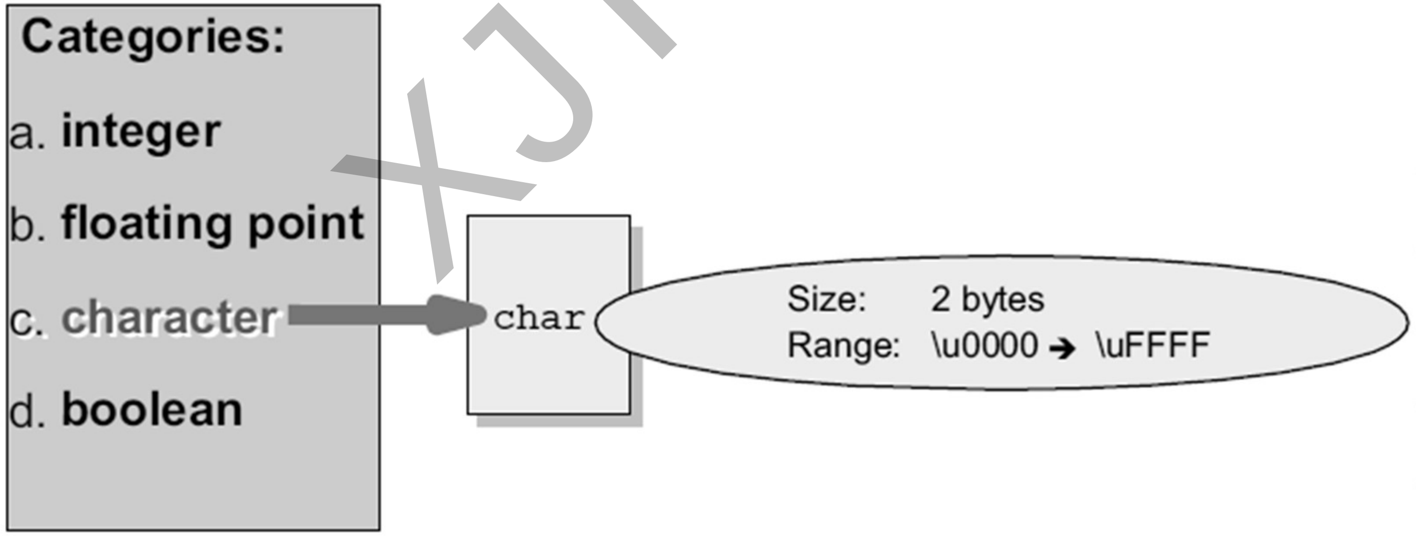
Primitives:floating points

- "General" numbers (can have fractional parts)
- Initialized to zero



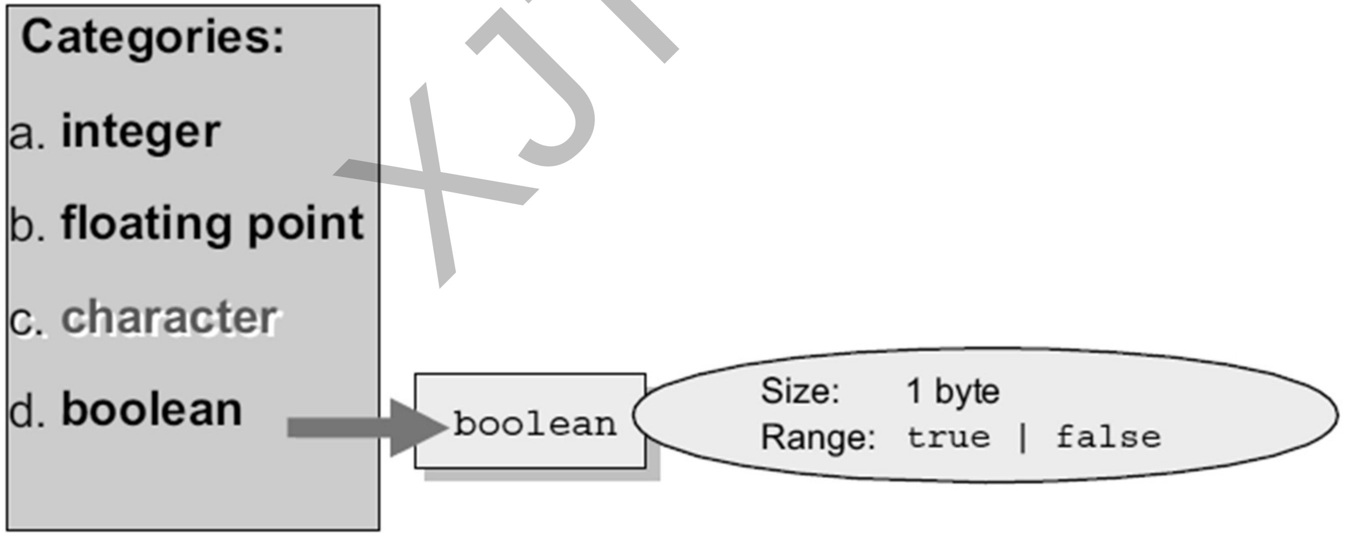
Primitives: characters

- Any unsigned Unicode character is a **char primitive** data type
- A **character** is a single Unicode character between two single quotes
- Initialized to zero (\u0000)



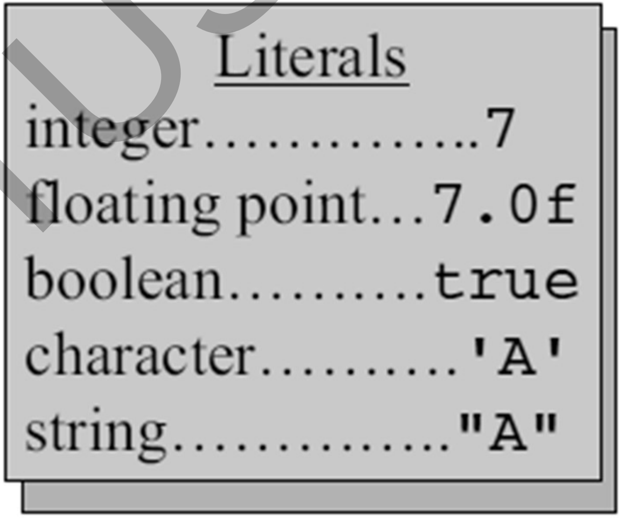
Primitives: booleans

- **boolean** values are distinct in Java
  - An **int** value can NOT be used in place of a **boolean**
  - A **boolean** can store either true or false
- Initialized to false



Literals

- A literal is a value
- Six kinds:
  - integer
  - floating point
  - boolean
  - character
  - String
  - null

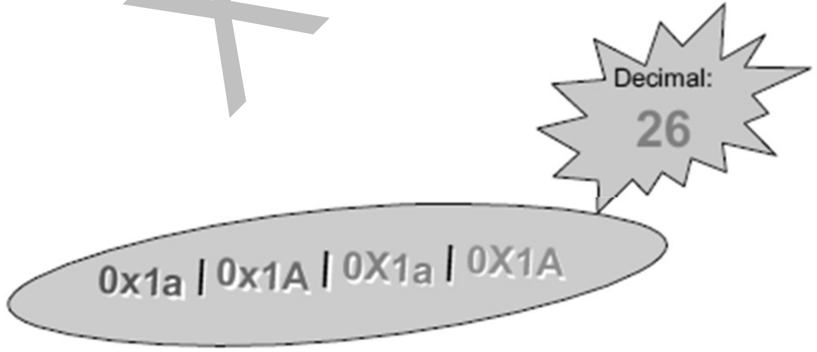


Literals

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**



Primitive literals: floating point

- float literals end with an f (or F)
  - 7.1f
- double literals end with a d (or D)
  - 7.1D
- 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 S
  - \b (backspace BS, Unicode \u0008)
  - \t (horizontal tab HT, Unicode \u0009)
  - \n (linefeed LF, Unicode \u000a)
  - \f (form feed FF, Unicode \u000c)
  - \r (carriage return CR, Unicode \u000d)
- Some characters may need to be escaped when used in string literals
  - \" (double quote “, Unicode \u0022)
  - \’ (single quote ’, Unicode \u0027)
  - \\ (backslash \, Unicode \u005c)
- Hexadecimal Unicode values can also be written ‘\uXXXX’

Casting primitive types

- 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
- Casting a value allows it to be treated as another type
- 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 versus explicit casting

- Casting is automatically done when no loss of information is possible
  - byte → short → int → long → float → double
- An explicit cast is 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?
```

Declarations and initialization

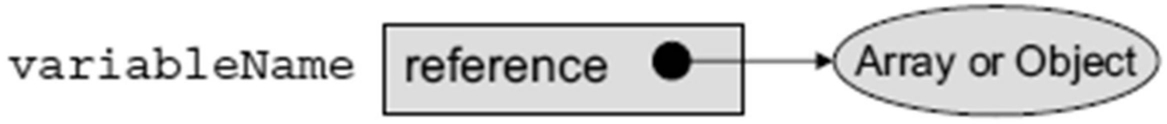
- Variables must be declared before they can be used
- Single value variables (variables that are not arrays) must be initialized before their first use in an expression
  - Declarations and initializations can be combined
  - Use **=** for assignment (including initialization)
- Examples:

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

Arrays

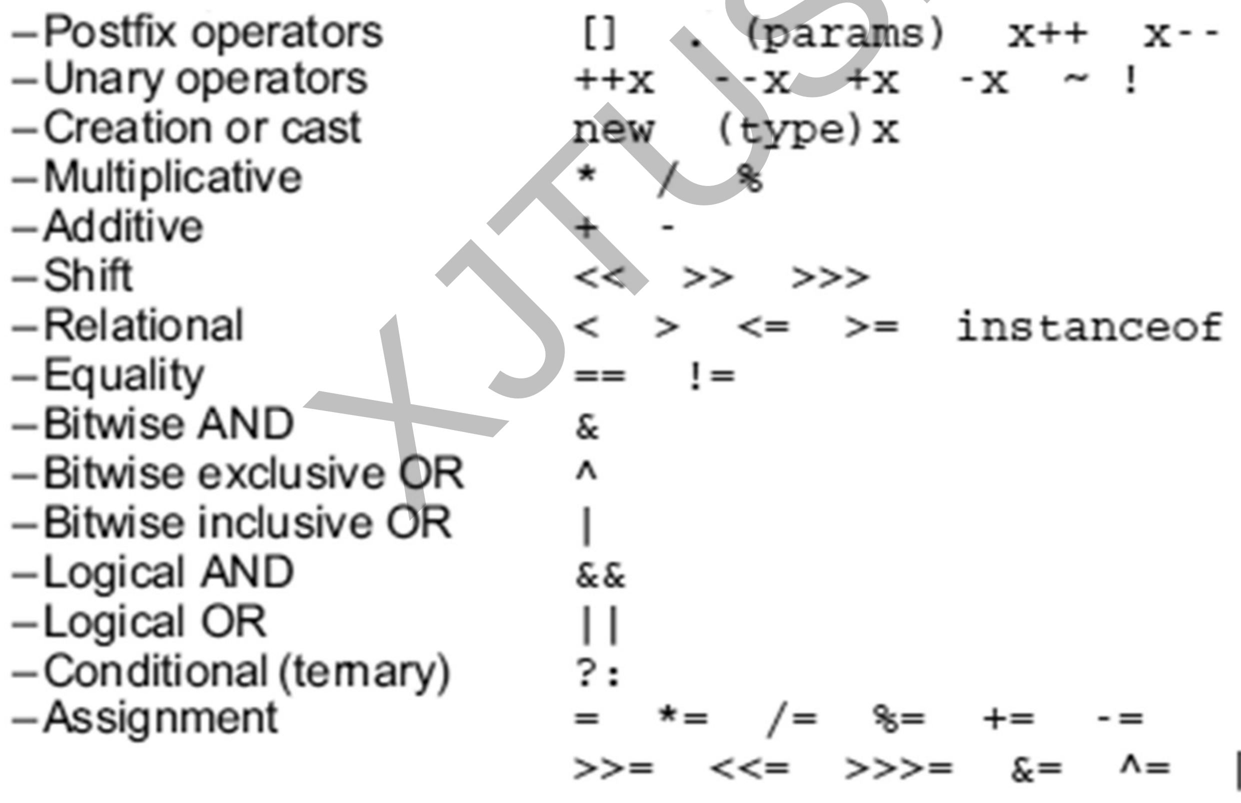
- Arrays must also be declared before use
  - Have fixed size
- May be specified by a literal, by an expression, or implicitly
  - May be optionally initialized
  - Have default values depending on their type
  - Are always zero-based (array[0] is the first element)
- Examples:

```
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 exclusive OR	`^`
–Bitwise inclusive OR	`|`
–Logical AND	`&&`
–Logical OR	`||`
–Conditional (temary)	`?:`
–Assignment	`=   *=   /=   %=   +=   -=`
	`>>=   <<=   >>>=   &=   ^=   |=`

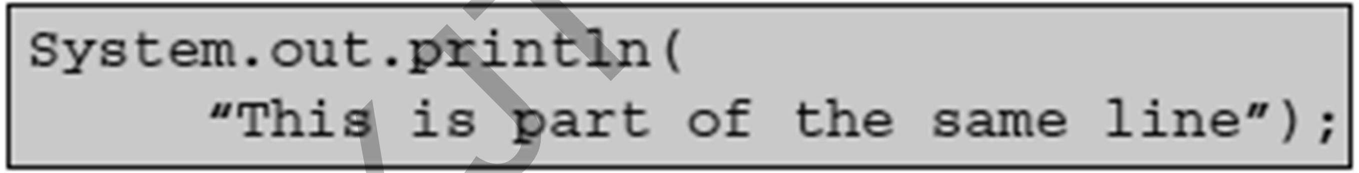
Comments

■ Java supports three kinds of comments:

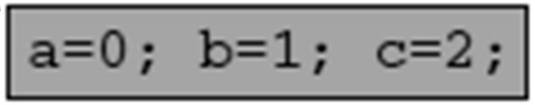
```
// The rest of the line is a comment
// No line breaks

/* Everything between
is a comment  */

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

Statements

- Statements are terminated by a semicolon
- Several statements can be written on one line
- A statement can be split over several lines



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



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

Checkpoint

- 1. What are the four types of Java integers?
- 2. What are the two types of Java floating point numbers?
- 3. What is the difference between a byte and a **char**?
- 4. When does Java provide implicit casts?
- 5. What are the three types of comments, and when would a developer use them?