JAVA PROGRAMMING BASICS

Module 1: Java Overview

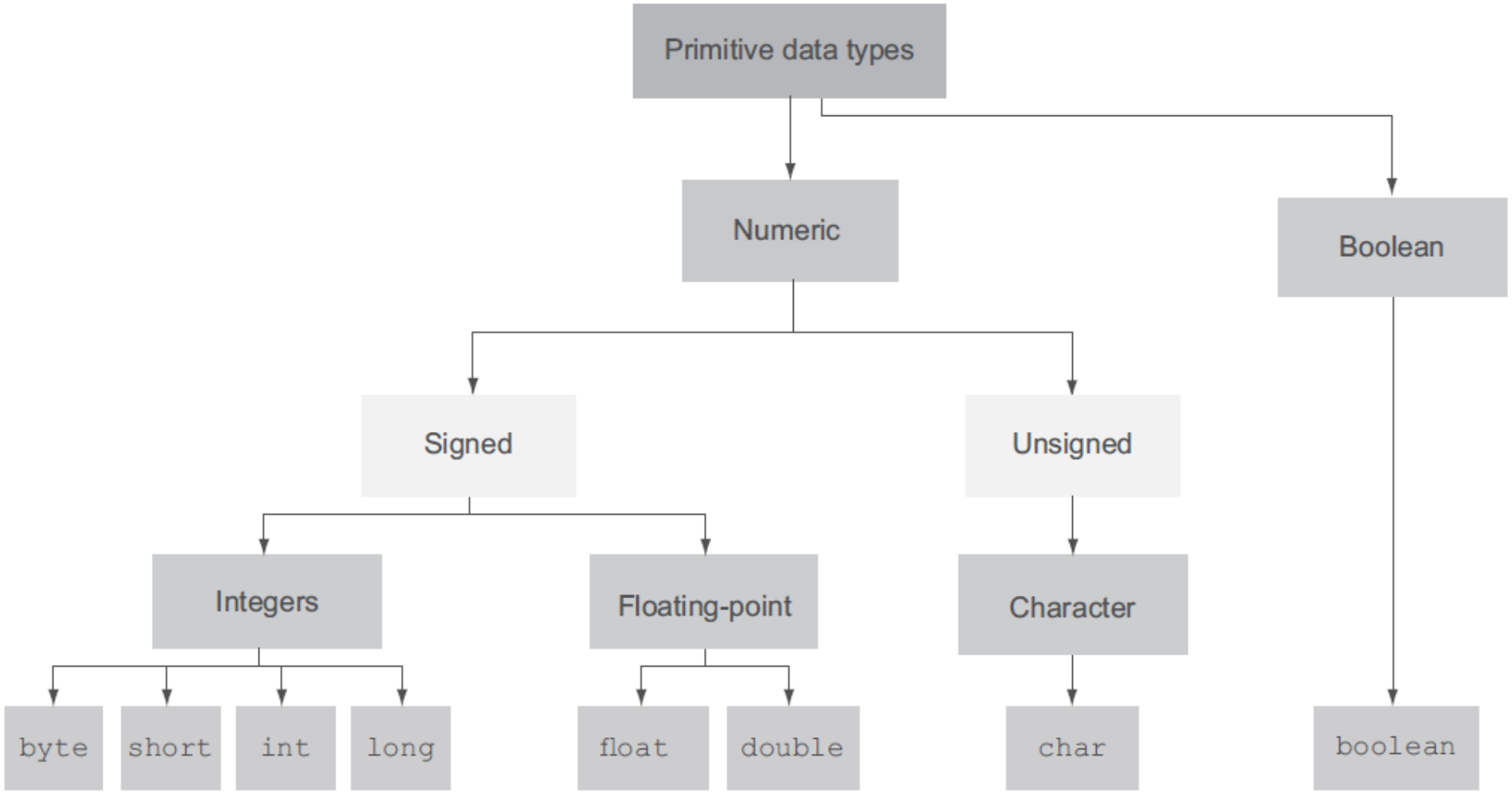
Training program

1. Java Fundamentals
2. Start programming with Java, create simple console application
3. Classification of Data Types
4. Primitive types in java
5. Control Flow Statements
6. Arrays

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Primitive types in Java

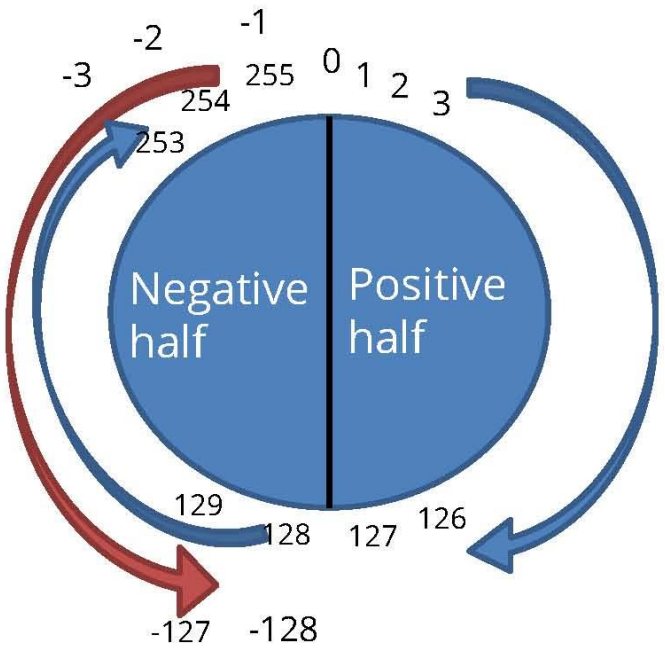


Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Signed and unsigned number representations

Unsigned numbers	Binary	Hex	Signed numbers
0	0000 0000	00	0
1	0000 0001	01	+1
2	0000 0010	02	+2
|	|	|	|
|	|	|	|
|	|	|	|
|	|	|	|
127	0111 1111	7F	+127
128	1000 0000	80	-128
129	1000 0001	81	-127
|	|	|	|
|	|	|	|
|	|	|	|
254	1111 1110	FE	-2
255	1111 1111	FF	-1



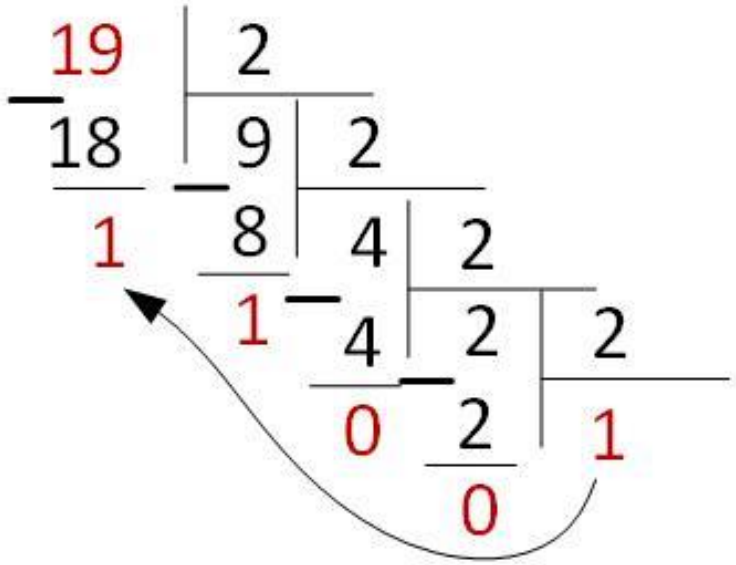
8 bit number system

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Integer to Binary Representation

$$19 = \overset{4\ 3\ 2\ 1\ 0}{10011} = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 19$$



Representation of unsigned integers in binary, octal and hexadecimal number formats 1/2

decimal	binary	octal	hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a

As with all numbering systems most significant digits are at left, least significant digits are at right.

Representation of unsigned integers in binary, octal and hexadecimal number formats 2/2

decimal	binary	octal	hexadecimal
11	1011	13	b
12	1100	14	c
13	1101	15	d
14	1110	16	e
15	1111	17	f
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Negative Integer to Binary Representation

$$19 = \overset{4\,3\,2\,1\,0}{10011}$$

$$-19 = -\overset{4\,3\,2\,1\,0}{10011}$$

$$\begin{array}{rl} & 01100 \\ + & \quad\;\; 1 \\ \hline -19 = & 01101 \end{array}$$

inversion

2-complement number representation

CHECK:

$-19 + 19 = 0$

$$\begin{array}{rl} & 01101 \\ + & 10011 \\ \hline & \cancel{1}00000 \end{array}$$

overflow

Twos-complement number representation

Number in decimal	Number in two`s complement binary
5	0000 0000 0000 0101
4	0000 0000 0000 0100
3	0000 0000 0000 0011
2	0000 0000 0000 0010
1	0000 0000 0000 0001
0	0000 0000 0000 0000
-1	1111 1111 1111 1111
-2	1111 1111 1111 1110
-3	1111 1111 1111 1101
-4	1111 1111 1111 1100
-5	1111 1111 1111 1011

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Floating Point Representation

Sign

Single Precision

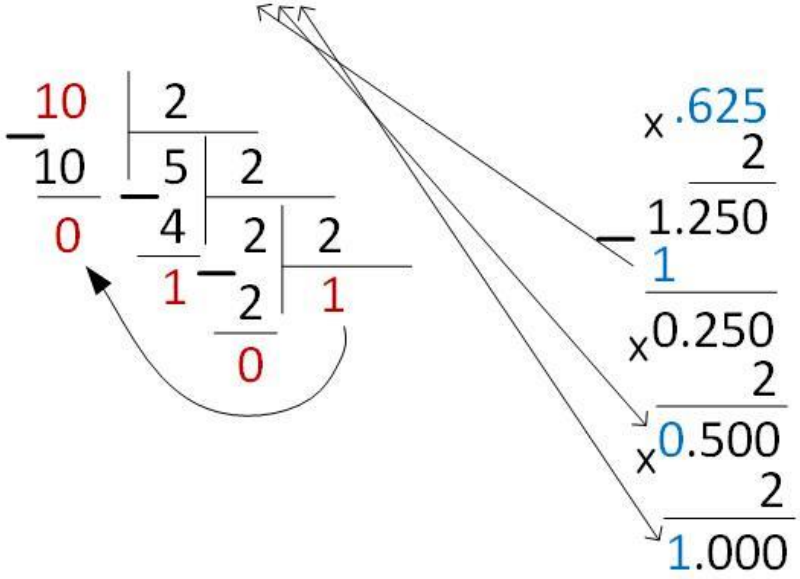
Sign

Double Precision

IEEE Standard for Floating-Point Arithmetic (IEEE 754)

Floating Point Representation

$$10.625 = 1010.101 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 10.625$$



$1010.101 = 1.010101 \cdot 2^3$

mantissa

exponent = 3 + 127 = 130 = 10000010

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

The integer types

Integer Length	Name or Type	Range
8 bits	byte	-2^7 to $2^7 - 1$
16 bits	short	-2^{15} to $2^{15} - 1$
32 bits	int	-2^{31} to $2^{31} - 1$
64 bits	long	-2^{63} to $2^{63} - 1$

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Unicode Character Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
0010	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
0020		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0070	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Basic Latin

Open in an individual page

Range: 0000–007F

Quantity of characters: 128

type: alphabet

Languages: english, german, french, italian, polish

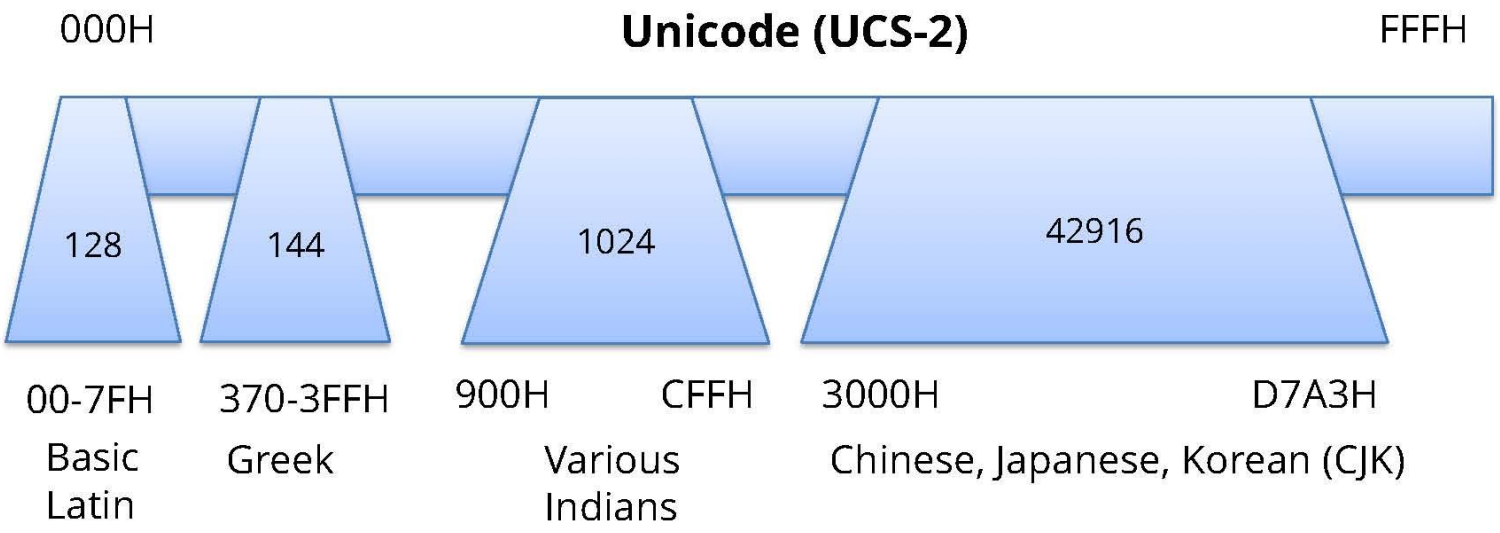
https://unicode-table.com/en/

The char type

```
char c1 = 'z';     //Use single quotes
char c2 = 122;  //Char code in decimal (z)
```

Basic Multilingual Plane (BMP).

Integer Length	Name or Type	Range
16 bits	char	'\u0000' (or 0) to '\uffff' (or 65,535).



Unicode 1000_H - 10FFFFH - *supplementary characters*

char values are **unsigned** integer values

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

The float and double types

Integer Length	Name or Type	Range
32 bits	float	-3.4E38 to 3.4E38
64 bits	double	-1.7E308 to 1.7E308

The float data type is a single-precision 32-bit IEEE 754 floating point

The double data type is a double-precision 64-bit IEEE 754 floating point

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

The boolean type

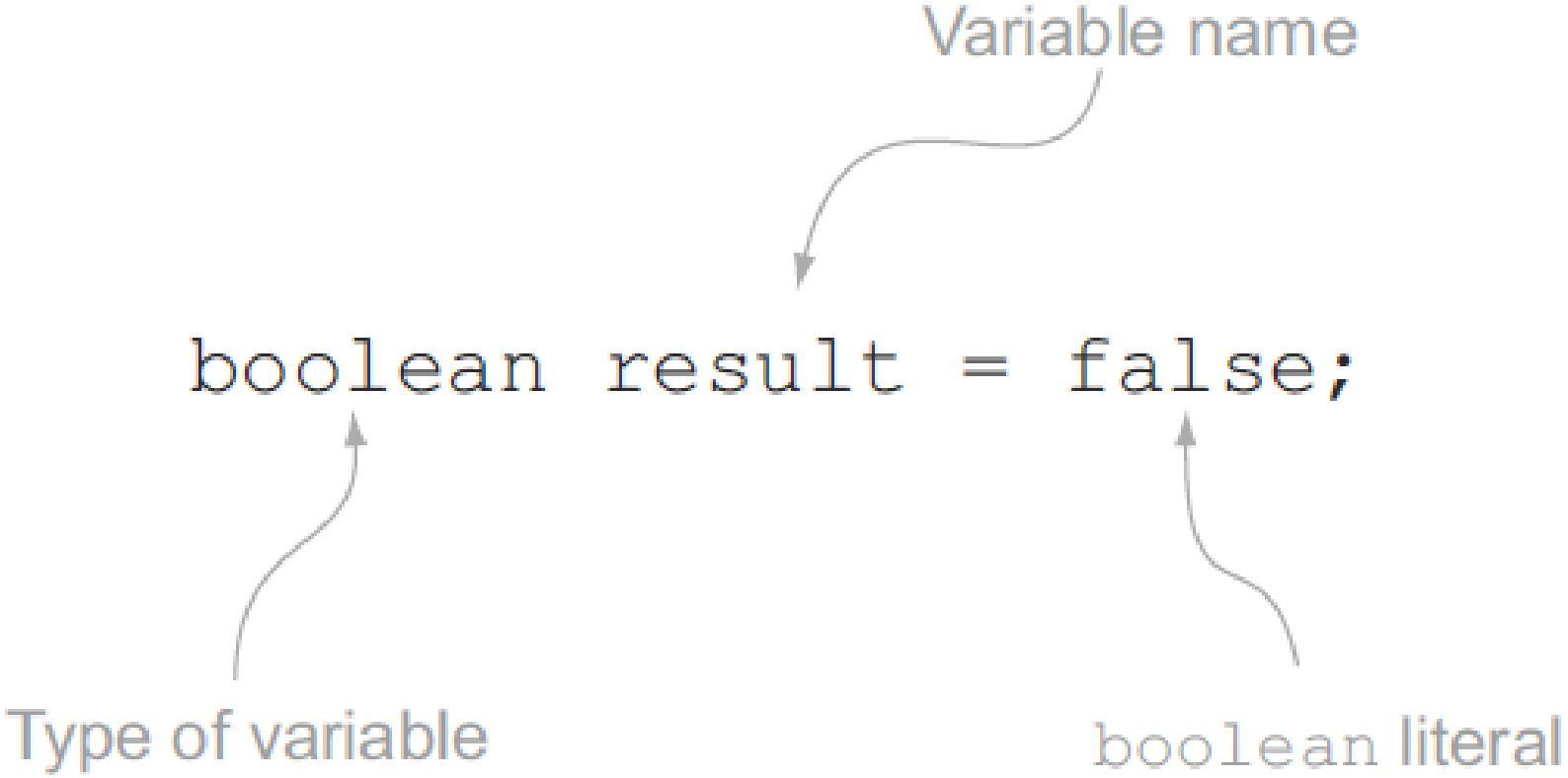
Integer Length	Name or Type	Range
-	boolean	true, false

The boolean data type has only two possible values: true and false

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Variable declaration

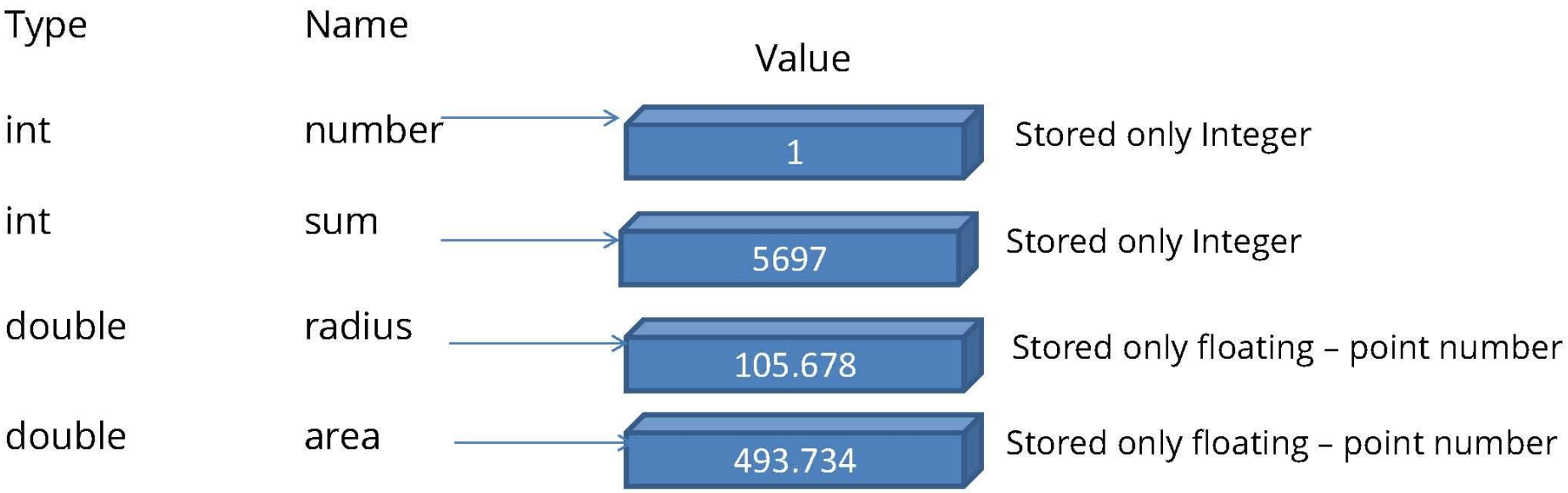


A ***literal*** is a fixed value that doesn't need further calculations in order for it to be assigned to any variable.

Java is a strongly typed language

Variables

A **variable** has a **name** and stores a **value** of the declared **type**



Identifiers

- Identifier is a name given to a variable, class or method or package
- Identifiers must start with a letter. The following characters can be digits.
- Identifiers are case sensitive.

- userName
- user_name
- _sys_var1
- $change

Literals 1/2

- Literal in Java refer to fixed values that do not change during the execution of the program
- Java supports several types of constants
- Integer Literal (prefixes: 0 - for octal, 0x - for hexadecimal, 0b - for binary,
- Real Literal (suffixes: F - for float) suffixes: L - for long)
- Character Literal (in ' ')
- String Literal (in " ")
- Backslash Literal (in '\ ')

```
long baseDecimal = 100_267_760_435L;    long hexVal = 0x10_BA_75;
long binVal = 0b1_0000_10_11;    long octVal = 0_4_13;
float floatLiteral = 1_00. 48F;
```

Java 7 introduced the use of underscores as part of the literal

Literals 2/2

1. **byte** b1 = 100;
2. **short** s2 = 1000;
3. **int** i3 = 20000;
4. **long** k4 = 2345678923456L;
5. **float** f5 = 18.456F;
6. **double** d6 = 77.234;
7. **char** c7 = **'a'**;
8. **boolean** b8 = **true**;

The suffix L is required when the value of the literal is greater than the maximum int

The suffix F is required for any float literal value

```
double incl = 1.201762e2;     //120.1762
```

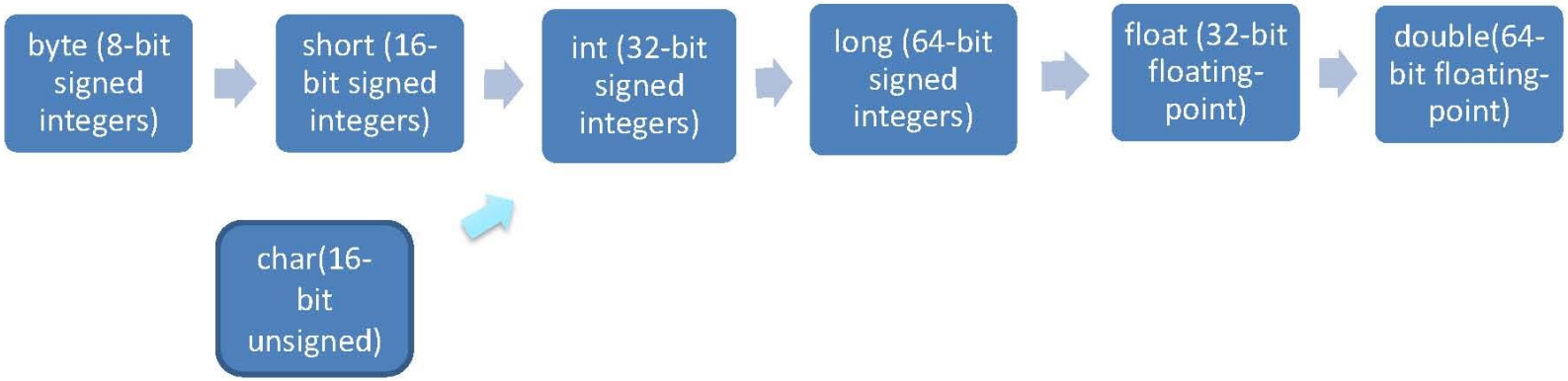
You can use a literal decimal value in scientific notation

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Type casting 1/2

Type casting: In computer science, type conversion, typecasting, and coercion are different ways of, implicitly or explicitly, changing an entity of one data type into another.



Implicit vs Explicit Casting

Implicit conversion results in automatic widening:

```
byte number = 10;
short bigNumber = number;          //implicit casting
System.out.println(number);        //10
System.out.println(bigNumber);     //10
```

On the contrary, **explicit casting** is a forceful conversion, which might result in loss of data:

```
short charCode = 336;              //character Ő ('\u0150')
byte by = (byte) charCode;         //explicit casting (0x50)
System.out.println(charCode);      //336
System.out.println(by);            //80
```

Type casting 2/2

```
1. long bigVal = 99L;
2. int x1 = bigVal;// Wrong, needs a cast
3. int x2 = (int) bigVal;// OK
4. int x3 = 99L;// Wrong, needs a cast
5. int x4 = (int) 99L;// OK, but loss of data
6. int x5 = 99;// default integer literal
```

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Scope of variables

Here are the available scopes of variables:

- Local variables (also known as method-local variables) (they also may be defined within code constructs such as if-else constructs, looping constructs, switch statements etc.
- Method parameters (also known as method arguments)
- Instance variables (also known as attributes, fields, and nonstatic variables)
- Class variables (also known as static variables)

The scope of a variable ends when the brackets of the block of code it's defined in get closed.

Variables with the same name in different scopes

- You can't define a static variable and an instance variable with the same name in a class:

```
class MyPhone {
   static boolean softKeyboard = true;
   boolean softKeyboard = true;        // Variable s already defined
}                                      // in the scope.
```

- Local variables and method parameters can't be defined with the same name:

```
void myMethod(int weight) {
   int weight = 10;                    // Variable s already defined
}                                      // in the scope.

...
```

Variables with the same name in different scopes

- A class can define local variables with the same name as the instance or class variables, also referred to as *shadowing*:

```
class MyPhone {
    static boolean softKeyboard = true;
    String phoneNumber;
    void myMethod() {
        boolean softKeyboard = true;          //Variable shadowing
        String phoneNumber;                   //Variable shadowing
    }
}
```

Local variable type inference - var keyword

- **Inference** is a capability of the Java compiler to determine the type of the local variable, by using the information that is already available in the code – like literal values, method invocations and their declaration
- The compiler infers the type using the information that is already available in the code and adds it to the bytecode it generates.

since Java 10

Local variable type inference - var keyword

```
public class VarKeyword {
//     var x = 5; //it isn't local variable
//     static var y = 6; //it isn't local variable
    {
        var name = "Aqua Blue";
    }
    static {
        var anotherLocalVar = 19876;
    }
    public static void main(String[] args) {
        var a = 2;              // int
        var b = 2.5;            // double
        var c = 'y';            // char
        var d = true;           // boolean
        var e = "Hi";           // String
```

...

since Java 10

Local variable type inference - var keyword

```
...
        var f = 2L;           // long
        var g = 2F;           // float
        var h = 2D;           // double
        var i = (short) 2; // short
        e = "Bye";
//        e = 5; //incompatible types: int can't
                 //be converted to String
        int[] arr = {1, 2, 3};
        for (var j = 0; j < 10; j++) {
            System.out.println(arr[j]);
        }
//        var nullVar = null; //Cannot infer type
                          // for local variable nullVar
        var nullVar = (String) null;
...
```

since Java 10

Local variable type inference - var keyword

```
...
//          var z; //Cannot use 'var' on variable
                   // without initializer
//          var[] arr = {1,2,3};  //'var' isn't
                              // allowed as an element type
                              // of an array

//          void someMethod(var a){ // Can't use
                                    // as parameter type
    }
}
```

since Java 10

Where you can and can’t use var

You can use var - for local variables:

- For the variables defined within both instance and static initializers;
- Within a method – both instance and static (including constructors);
- For the variables defined in control statements like: if-else, loops (for, while, do), switch statements etc;
- Within try-with-resources statement;

You can't use var:

- For static and instance variables (fields), for arrays;
- For method parameter types, return types or to the variable defined with catch handlers.
- For variable that is not initialized or equals null.

since Java 10

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Brief overview of operators in java

Operators	Associative
++ -- + unary - unary ~ ! (<data_type>)	R to L
* / %	L to R
+ -	L to R
<< >> >>>	L to R
< > <= >= instanceof	L to R
== !=	L to R
&	L to R
^	L to R
|	L to R
&&	L to R
||	L to R
<boolean_expr> ? <expr1> : <expr2>	R to L
= *= /= %= += -= <<= >>>= &= ^= |=	R to L

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Main arithmetic operators

Operator	Name	Example expression	Meaning
*	Multiplication	a * b	a times b
/	Division	a / b	a divided by b
%	Remainder (modulus)	a % b	a the remainder dividing a by b
+	Addition	a + b	a plus b
_	Subtraction	a - b	a minus b

```
char ch1 = '1';  //49
char ch2 = '2';  //50
System.out.println(ch1 + ch2);  //99

System.out.println(ch1 & ch2);  //49 & 50 = 00110001 & 00110010 = 00110000 = 48
```

We can use char as operands.
It is interpreted as integers.

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Increments and decrements

Expression	Process	Example	End Result
i++	Add 1 to a variable after use	int i=10,x; x=i++	i=11 x=10
++i	Add 1 to a variable before use	int i=10,x; x=++i;	i=11 x=11
i--	Subtract 1 from a variable after use	int i=10,x; x=i--;	i=9 x=10
--i	Subtract 1 from a variable before use	int i=10,x; x=--i;	i=9 x=9

Module contents

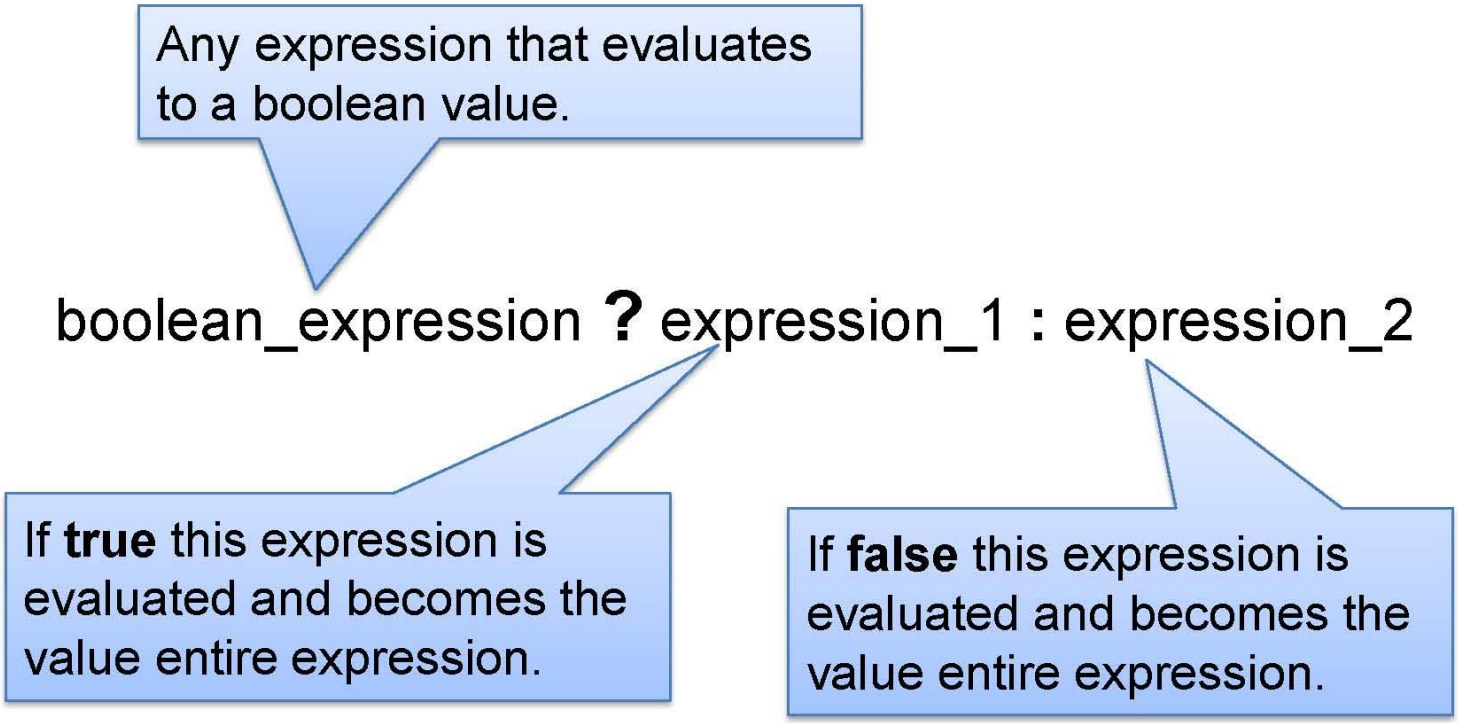
- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Relational operators

Operator	Name	Example Expression	Meaning
==	test for equality	a == 0	Is a equal to 0
!=	test for inequality	s != null	Is s not equal to null
<	less than	b < c	Is b value less than c value
>	greater than	d > 5	Is d value less than 5
<=	less than or equal	e <= 0	Is e value less or equal than 0
>=	greater than or equal	f >= 0	Is f value greater than or equal to 0

All relational operators returns boolean type value - true or false

Ternary operator




```
boolean cond = true;
int x = cond? 25:17;
```

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Logical operators

A	B	A|B	A&B	A^B	!A
false	false	false	false	false	true
true	false	true	false	true	false
false	true	true	false	true	true
true	true	true	true	false	false

| the OR operator
& the AND operator
^ the XOR operator
! the NOT operator

We use logical operators for composite relational operators,
e.g. (a > 0) | (a % 2 == 0),
(s == t) & (s != null)

Short-Circuit Logical Operators

	Meaning	Short circuit?
&&	and	yes
&	and	no
||	or	yes
|	or	no

If you use the || and && forms, rather than the | and & forms of these operators, Java will not bother to evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone.

We use rather short circuit logical operators for composite relational operators, e.g. (a > 0) **||** (a % 2 == 0),

(s == t) **&&** (s != null)

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Bitwise Operators

```
int a = 10; // 00001010 = 10
int b = 12; // 00001100 = 12
```

&
AND

```
a          00000000000000000000000000001010    10
b          00000000000000000000000000001100    12
a & b      00000000000000000000000000001000     8
```

|
OR

```
a          00000000000000000000000000001010    10
b          00000000000000000000000000001100    12
a | b      00000000000000000000000000001110    14
```

^
XOR

```
a          00000000000000000000000000001010    10
b          00000000000000000000000000001100    12
a ^ b      00000000000000000000000000000110     6
```

~
NOT

```
 a         00000000000000000000000000001010    10
~a         11111111111111111111111111110101   -11
```

Bit shift operators 1/2

```
int a =  3; // ...00000011 =  3
int b = -4; // ...11111100 = -4
```

<< Left

```
a        00000000000000000000000000000011     3
a << 2   00000000000000000000000000001100    12

b        11111111111111111111111111111100    -4
b << 2   11111111111111111111111111110000   -16
```

>> Right

```
a        00000000000000000000000000000011     3
a >> 2   00000000000000000000000000000000     0

b        11111111111111111111111111111100    -4
b >> 2   11111111111111111111111111111111    -1
```

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Bit shift operators 2/2

```
int a =  3; // ...00000011 =  3
int b = -4; // ...11111100 = -4
```

>>>
Right 0

```
a          00000000000000000000000000000011      3
a >>> 2    00000000000000000000000000000000      0

b          11111111111111111111111111111100     -4
b >>> 2    00111111111111111111111111111111     +big
```

Module contents

- Primitive types in Java
  - Signed and unsigned number presentation
  - Representation of entire unsigned integers in bin, oct, hex number formats
  - Twos-complement number representation
  - Floating Point Number Representation
  - The integer types
  - The char type
  - The float and double types
  - The boolean type
  - Variables and identifiers
  - Type Casting
  - Scope of variables
  - Brief overview of operators in Java
  - Main arithmetic operators
  - Increments and decrements
  - Relational operators, Ternary operator
  - Logical operators, Short-Circuit Logical operators
  - Bit wise operators
  - Bit shift operators
  - Operators precedence

Operator Precedence 1/4

Precedence	Operator	Description	Association
1	.	Member	L to R
	()	Function call	L to R
	[]	Array element reference	L to R
2	++,--	Postincrement, Postdecrement	R to L
3	++,--	Preincrement, Predecrement	R to L
	+,-	Unary plus, unary minus	R to L
	~	Bitwise compliment	R to L
	!	Boolean NOT	R to L

The operator on top has the highest precedence, and operators within the same group have the same precedence and are evaluated from left to right

Operator Precedence 2/4

Precedence	Operator	Description	Association
4	new	Create object	R to L
	(type)	Type cast	R to L
5	*,/,%	Multiplication, division, remainder	L to R
6	+,-	Addition, subtraction	L to R
	+	String concatenation	L to R

Operator Precedence 3/4

Precedence	Operator	Description	Association
7	<<, >>, >>>	Shift operator	L to R
8	<, <=, >, >=	Less than, Less than or equal to, greater than, greater than or equal to	L to R
	instanceof	Type comparison	L to R
	==, !=	Value equality and inequality	L to R
	==, !=	Reference equality and inequality	
10	&	Boolean AND	L to R
	&	Bitwise AND	

Operator Precedence 4/4

Precedence	Operator	Description	Association
11	^	Boolean XOR	L to R
12	^	Bitwise XOR	L to R
	|	Boolean OR	L to R
13	|	Bitwise OR	L to R
14	&&	Conditional AND	L to R
15	||	Conditional OR	L to R
	?:	Conditional Ternary Operator	L to R
16	=, +=, -=, *=, / =, %=, &=, ^=, | =, <<=, >> =, >>>=	Assignment Operators	R to L