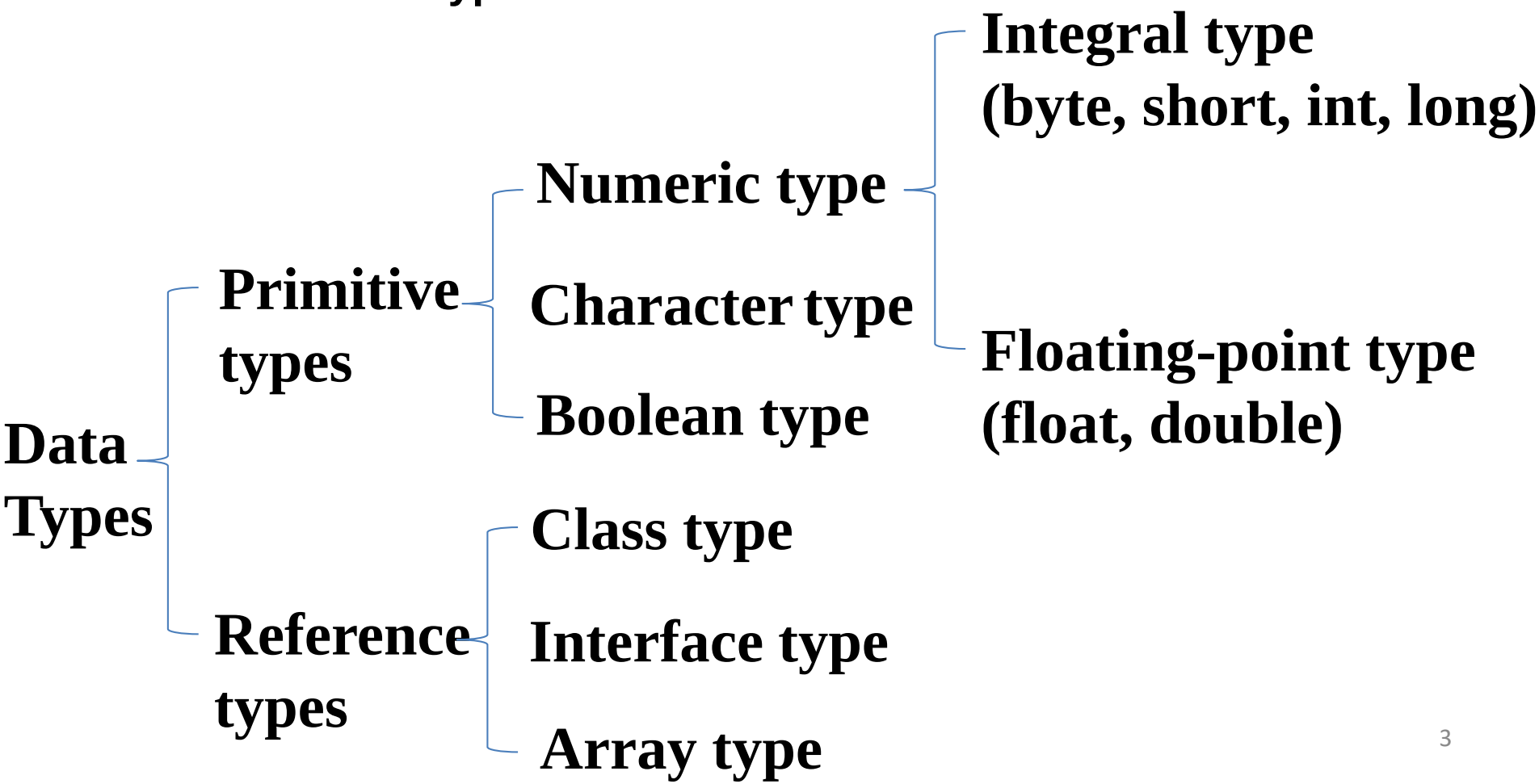
Java Data Types & Operators

Outline

- **Java data type**
- Operators

Java data types

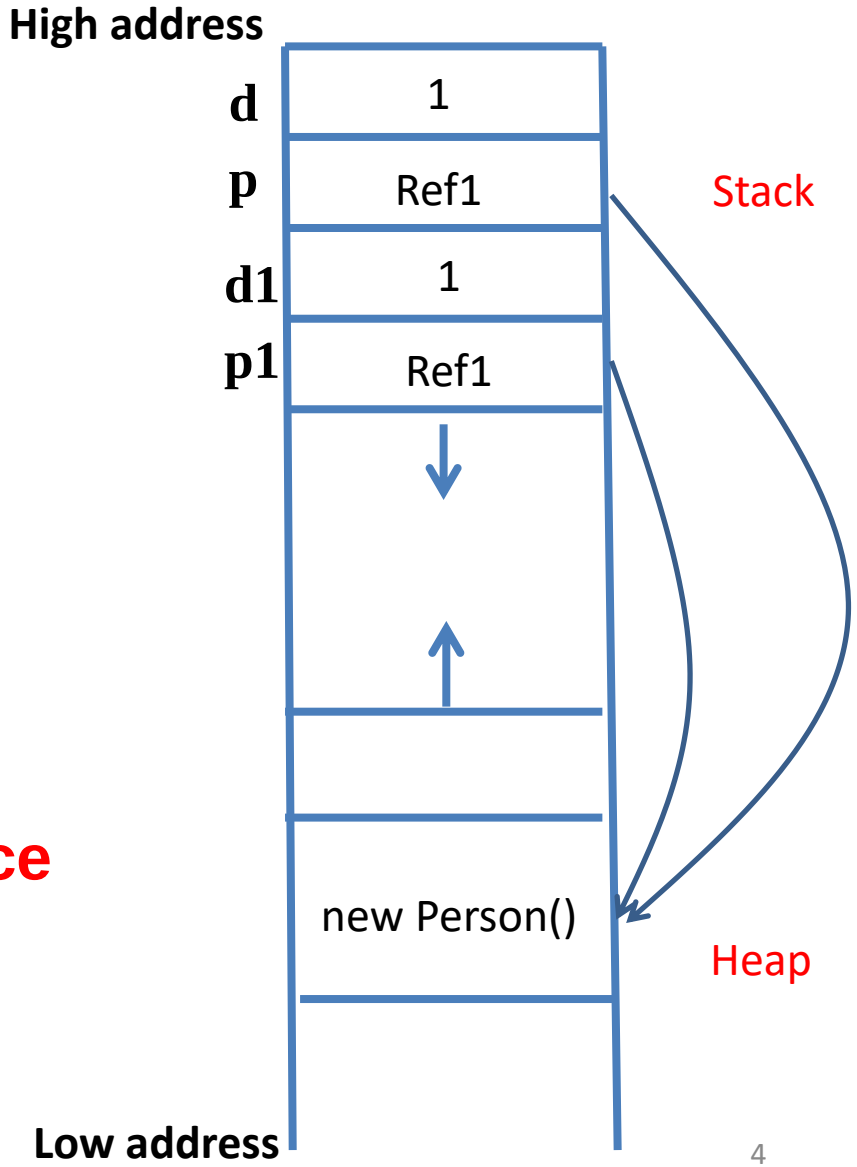
- **Java data types**
  - Primitive types
  - Reference types



Difference: Primitive and Reference Type

- Primitive type: stack
- Reference type: heap
  - double d = 1;
  - Person p = new Person();

- Assignment
  - double d1 = d; value
  - Person p1 = p; reference



Primitive type

IEEE 754

- $s \cdot m \cdot 2^{(e - N + 1)}$
- s is +1 or -1
- m is a positive integer less than 2^N integer N (N - float:24, double:53)
- e is an integer between $E_{min} = -(2^{K-1}-2)$ and $E_{max} = 2^{K-1}-1$ (K - float:8, double:11)

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	—	—	—	Boolean
char	16 bits	Unicode 0	Unicode 2^{16}- 1	Character
byte	8 bits	-128	+127	Byte
short	16 bits	-2^{15}	$+2^{15}-1$	Short
int	32 bits	-2^{31}	$+2^{31}-1$	Integer
long	64 bits	-2^{63}	$+2^{63}-1$	Long
float	32 bits	IEEE754	IEEE754	Float
double	64 bits	IEEE754	IEEE754	Double
void	—	—	—	Void

Primitive type

- **“wrapper” classes for the primitive data types allow you to make a non-primitive object on the heap to represent that primitive type**
- **Example**
  - char c = ‘x’;
  - Character ch = new Character(c); or
  - Character ch = new Character(‘x’);
- **Java SE5 autoboxing will automatically convert from a primitive to a wrapper type**
  - Character ch = ‘x’;
  - char c = ch;

Default values for primitive members

- ***Class Variables*** **- *a member of a class***
  - **Initialized using default values**
- ***Local Variables*** **- not fields of a class**
  - **Get some arbitrary value**

Primitive type	Default
boolean	false
char	'\uoooo' (null)
byte	(byte)o
short	(short)o
int	o
long	oL
float	o.of
double	o.od

Boolean type

- Represents a logical quantity with two possible values
  - indicated by the literals true and false
- Boolean expressions determine the control flow
  - if, while, do-while, for statements
  - If(a=1) cannot be used in Java
- Example

```
boolean  b = false;
if(b==true)  {
            //do something
}
```

Character type

- Indicate a single character value
- Size of each char is 16 bits (two bytes)
  - Support Unicode characters
- Character literals
  - Single- quoted character
    - char a = 'A';
  - Unicode sequence
    - char a1 = '\u0061';
    - Hexadecimal (base 16), denoted by a leading 0x or 0X followed by 0-9 or a-f
  - An escape character
    - char a2 = '\n';
    - A character preceded by a *backslash* (\) is an *escape sequence* and has special meaning to the compiler

Escape Sequences

Escape Sequence	Description
\uhhhh	Unicode character with hex representation **0xhhhh**
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

Integer type

- The integer types in Java are *byte*, *short*, *int*, and *long*
- Integer literals
  - Decimal (base 10) notation
    - 12, 314, etc.
  - Octal (base 8) notation
    - Denoted by a leading 0 in the number and digits from 0-7, e.g., 012
  - Hexadecimal (base 16) notation
    - denoted by a leading 0x or 0X followed by 0-9 or a-f, e.g., 0x12 or 0X12
  - Binary (base 2) notation (Java 7)
    - Denoted by a leading 0b or 0B in the number and digits from 0 or 1, e.g., 0b000100010

Integer type (Cont.)

- **Integer literals**
  - **Integer literals are of type int unless they are suffixed with an l or L**
  - **Long value**
    - **long i = 3; // 3 is converted from type int**
    - **long i = 3L;**
    - **long i = 3l;**
  - **Avoid to use the lowercase letter l, since it looks like the number 1**

Floating-point type

- The float-pointing types in Java are *float* and *double*
- Float-pointing literals
  - Decimal (base 10) notation
    - 3.14, 314.0, .314, etc.
      - Underscore character for format, e.g., 123_456.789_000 (Java 7)
  - Scientific notation
    - 3.14e2 3.14E2 3.14e-2
    - e/E means 10 to the power rather than 2.718
  - Floating-point literals are of type double unless they are suffixed with an f or F
    - double d = 3.14;
    - float f = 3.14f;

Literals

```
1  //: operators/Literals.java
2  import static net.mindview.util.Print.*;
3
4  public class Literals {
5    public static void main(String[] args) {
6      int i1 = 0x2f; // Hexadecimal (lowercase)
7      print("i1: " + Integer.toBinaryString(i1));
8      int i2 = 0X2F; // Hexadecimal (uppercase)
9      print("i2: " + Integer.toBinaryString(i2));
10     int i3 = 0177; // Octal (leading zero)
11     print("i3: " + Integer.toBinaryString(i3));
12     char c = 0xffff; // max char hex value
13     print("c: " + Integer.toBinaryString(c));
14     byte b = 0x7f; // max byte hex value
15     print("b: " + Integer.toBinaryString(b));
16     short s = 0x7fff; // max short hex value
17     print("s: " + Integer.toBinaryString(s));
18     long n1 = 200L; // long suffix
19     long n2 = 200l; // long suffix (but can be confusing)
20     long n3 = 200;
21     float f1 = 1;
22     float f2 = 1F; // float suffix
23     float f3 = 1f; // float suffix
24     double d1 = 1d; // double suffix
25     double d2 = 1D; // double suffix
26     // (Hex and Octal also work with long)
27   }
28 }
```

Output:

i1: 101111

i2: 101111

i3: 1111111

c: 1111111111111111

b: 1111111

s: 1111111111111111

Identifier

- Your variables, method, class names, etc.
- Syntax for valid Java identifiers
  - Must have at least one character
  - Identifiers contain alphabetic characters, digits, underscore (_), and dollar sign ($)
  - The first character ***cannot be a digit***
- Coding style
  - Case sensitive
  - Camel-casing
  - Capitalize the first letter of a class name
  - For almost everything else—methods, fields (member variables), and object reference names
    - the first letter of the identifier is lowercase

Outline

- Java data type
- **Operators**

Operators

- Mathematical operators
  - +, -, *, /, %, ++, --
- Relational operators
  - >, <, >=, <=, ==, !=
- Logical operators
  - ! , &&, ||
- Bitwise operators
  - &, |, ^, ~
- Shift operators
  - >>, <<, >>>
- Assignment
  - =

Mathematical Operators

- +, -, *, /, %, -, ++, --,
- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
  - 15/4 15.0/2
  - Integer division truncates the result rather than rounds
- Modulus (%)
  - Produce the remainder from integer division
  - 100%3 a%2 a%10

Mathematical Operators (Cont.)

- Unary minus and plus operators
  - x = -a; x = a * (-b);
- Auto increment and decrement
  - Pre-increment/decrement, e.g., ++a, --a
    - Operation is performed and the value is produced
  - Post-increment/decrement, e.g., a++, a--
    - The value is produced and then operation is performed

```
1  //: operators/AutoInc.java
2  // Demonstrates the ++ and -- operators.
3  import static net.mindview.util.Print.*;
4
5  public class AutoInc {
6    public static void main(String[] args) {
7      int i = 1;
8      print("i : " + i);
9      print("++i : " + ++i); // Pre-increment
10     print("i++ : " + i++); // Post-increment
11     print("i : " + i);
12     print("--i : " + --i); // Pre-decrement
13     print("i-- : " + i--); // Post-decrement
14     print("i : " + i);
15   }
16 }
```

Relational Operators

- Less than (<), greater than (>), Less than or equal to (<=), greater than or equal to (>=)
  - Won't work with type ***boolean***
- Equivalent (==), not equivalent (!=)
  - Work with all primitives
  - Work with all objects, but may confuse you

```
1  //: operators/Equivalence.java
2
3  public class Equivalence {
4    public static void main(String[] args) {
5      Integer n1 = new Integer(47);
6      Integer n2 = new Integer(47);
7      System.out.println(n1.equals(n2));
8
9    }
10 }
```

Relational Operators (Cont.)

- The default behavior of equals() is to compare reference
  - Unless you ***override*** ***equals()*** in your new class
- Most of the Java library classes implement ***equals()***
  - Compare the content of objects

```
1  //: operators/EqualsMethod2.java
2  // Default equals() does not compare contents.
3
4  class Value {
5    int i;
6  }
7
8  public class EqualsMethod2 {
9    public static void main(String[] args) {
10     Value v1 = new Value();
11     Value v2 = new Value();
12     v1.i = v2.i = 100;
13     System.out.println(v1.equals(v2));
14   }
15 }
```

Logical Operators

- **ADD (&&), OR (||), NOT (!)**
  - **Applied to *boolean values only***

- **Short-circuiting**
  - **The expression will be evaluated only *until* the truth or falsehood of the entire expression can be unambiguously determined**
  - **The latter parts of a logical expression might not be evaluated**

```
if ((d!=null) && (d.day >31)) {
        //do something with d
}

if(i <0 || i >31 ) {
        //do something
}
```

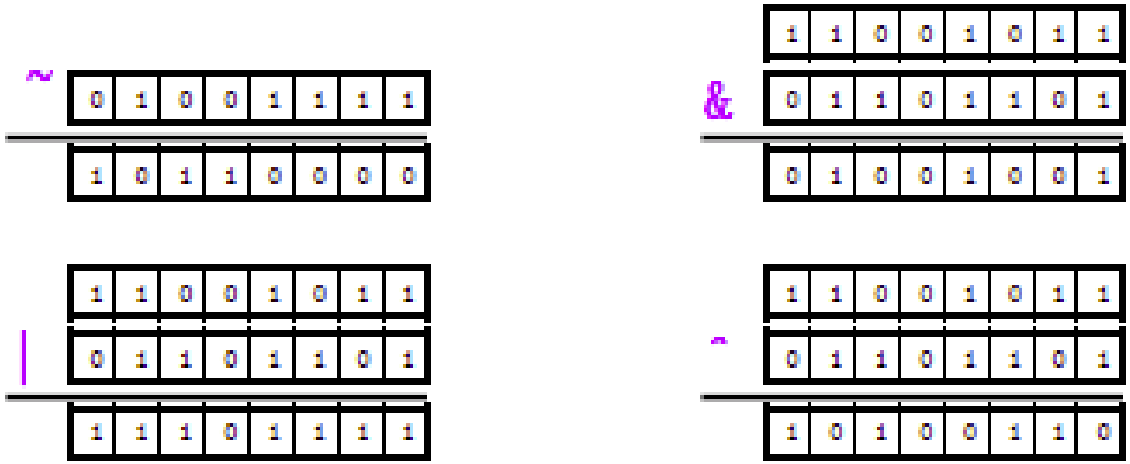
Bitwise Operators

- Bitwise ADD (&)
  - Produce a one if both input bits are one; otherwise, it produces a zero
- Bitwise OR (|)
  - Produce a one if either input bit is a one and produces a zero only if both input bits are zeros
- Bitwise Exclusive OR or XOR (^)
  - Produce a one if one or the other input bit is a one, but not both
- Bitwise NOT (~)
  - Unary operator
  - Produce the opposite of the input bit
- &=, |=, ^= are legitimate
- Boolean type can be treated as a one-bit value
  - Bitwise AND, OR, and XOR can be applied to boolean type

Bitwise Operators (Cont.)

- **Example**

a	b	!a	a&b	a | b	a^b	a&&b	a | | b
true	true	false	true	true	false	true	true
true	false	false	false	true	true	false	true
false	true	true	false	true	true	false	true
false	false	true	false	false	false	false	false



Shift Operators

- **Left-shift operator (<<)**
  - **Insert zeros at the lower-order bits**
- **Signed right-shift operator (>>)**
  - **If the value is positive, insert zeros at the higher – order bits**
  - **Otherwise, insert ones at the higher-order bits**
- **Unsigned right-shift operator (>>>)**
  - **Zeros are inserted at the higher-order bits**
  - **This operator does not exist in C or C++**
- **Applied to *char*, *byte*, *short*, *int*, or *long***
- **Shift a *char*, *byte*, or *short*, it will be promoted to *int***
- **<<=, >>=, >>>= are legitimate**

Shift Operators (Cont.)

Example

2227 =	00000000	00000000	00001000	10110011
2227<<3 =	00000000	00000000	01000101	10011000
2227>>3 =	00000000	00000000	00000001	00010110
2227>>>3 =	00000000	00000000	00000001	00010110
-2227 =	11111111	11111111	11110111	01001101
-2227<<3 =	11111111	11111111	10111010	01101000
-2227>>3 =	11111111	11111111	11111110	11101001
-2227>>>3 =	00011111	11111111	11111110	11101001

Assignment

- **Assignment (=)**
  - **Take the value of the right-hand side (*rvalue*) and copy it into the left-hand side (*lvalue*)**
  - **lvalue must be a distinct, named variable**
  - **rvalue is any constant, variable, or expression that produces a value**

```
1  //: operators/Assignment.java
2  // Assignment with objects is a bit tricky.
3  import static net.mindview.util.Print.*;
4
5  class Tank {
6    int level;
7  }
8
9  public class Assignment {
10   public static void main(String[] args) {
11     Tank t1 = new Tank();
12     Tank t2 = new Tank();
13     t1.level = 9;
14     t2.level = 47;
15     print("1: t1.level: " + t1.level +
16           ", t2.level: " + t2.level);
17     t1 = t2;
18     print("2: t1.level: " + t1.level +
19           ", t2.level: " + t2.level);
20     t1.level = 27;
21     print("3: t1.level: " + t1.level +
22           ", t2.level: " + t2.level);
23   }
24 }
```

```
/* Output:
1: t1.level: 9, t2.level: 47
2: t1.level: 47, t2.level: 47
3: t1.level: 27, t2.level: 27
*///:~
```

Assignment (Cont.)

- **Aliasing**
  - **Aliasing during method calls**

```
1  //: operators/PassObject.java
2  // Passing objects to methods may not be
3  // what you're used to.
4  import static net.mindview.util.Print.*;
5
6  class Letter {
7    char c;
8  }
9
10 public class PassObject {
11   static void f(Letter y) {
12     y.c = 'z';
13   }
14   public static void main(String[] args) {
15     Letter x = new Letter();
16     x.c = 'a';
17     print("1: x.c: " + x.c);
18     f(x);
19     print("2: x.c: " + x.c);
20   }
21 }
```

/* Output:
1: x.c: a
2: x.c: z
***///:~**

Casting operators

- Cast is used in the sense of “casting into a mold”
- Narrowing conversion
  - You run the risk of losing information
- Widening conversion
  - Casting is safe
  - An explicit cast is not needed
  - The new type will more than hold the information fro the old type
- *Boolean* does not allow any casting at all

Casting operators (Cont.)

- **Example**

```
1  //: operators/Casting.java
2
3  public class Casting {
4    public static void main(String[] args) {
5      int i = 200;
6      long lng = (long)i;
7      lng = i; // "Widening," so cast not really required
8      long lng2 = (long)200;
9      lng2 = 200;
10     // A "narrowing conversion":
11     i = (int)lng2; // Cast required
12   }
13 } ///:~
14
```

Casting operators (Cont.)

- **Truncation and rounding**
  - **Cast from a float or double to an integral value always truncate the number**

```
1  //: operators/CastingNumbers.java
2  // What happens when you cast a float
3  // or double to an integral value?
4  import static net.mindview.util.Print.*;
5
6  public class CastingNumbers {
7    public static void main(String[] args) {
8      double above = 0.7, below = 0.4;
9      float fabove = 0.7f, fbelow = 0.4f;
10     print("(int)above: " + (int)above);
11     print("(int)below: " + (int)below);
12     print("(int)fabove: " + (int)fabove);
13     print("(int)fbelow: " + (int)fbelow);
14   }
15 }
```

```
/* Output:
(int)above: 0
(int)below: 0
(int)fabove: 0
(int)fbelow: 0
*///:~
```

Casting operators (Cont.)

- **Truncation and rounding**
  - **Use *round()* methods in *java.lang.Math***

```
1  //: operators/RoundingNumbers.java
2  // Rounding floats and doubles.
3  import static net.mindview.util.Print.*;
4
5  public class RoundingNumbers {
6    public static void main(String[] args) {
7      double above = 0.7, below = 0.4;
8      float fabove = 0.7f, fbelow = 0.4f;
9      print("Math.round(above): " + Math.round(above));
10     print("Math.round(below): " + Math.round(below));
11     print("Math.round(fabove): " + Math.round(fabove));
12     print("Math.round(fbelow): " + Math.round(fbelow));
13   }
14 }
```

```
/* Output:
Math.round(above): 1
Math.round(below): 0
Math.round(fabove): 1
Math.round(fbelow): 0
*///:~
```

Promotion

- Perform any mathematical or bitwise operations on *char*, *byte* or *short*
  - Values will be prompted to *int*
  - The result will be *int*
- If you want to assign back into the smaller type, you must use a cast
  - You might be losing information
- The *largest data type* in an expression is the one that determines the size of the result of that expression
  - Multiply a *float* and a *double*, the result will be *double*
  - Add an *int* and a *long*, the result will be *long*

Ternary if-else operators

- **Conditional operator**
  - **Has three operands**
- **Expression**
  - **boolean-exp ? value0 ： value1**
  - **True, value0 is evaluated and the result become the value**
  - **False, value1 is evaluated and the result become the value**

```
1  //: operators/TernaryIfElse.java
2  import static net.mindview.util.Print.*;
3
4  public class TernaryIfElse {
5    static int ternary(int i) {
6      return i < 10 ? i * 100 : i * 10;
7    }
8    static int standardIfElse(int i) {
9      if(i < 10)
10       return i * 100;
11     else
12       return i * 10;
13   }
14   public static void main(String[] args) {
15     print(ternary(9));
16     print(ternary(10));
17     print(standardIfElse(9));
18     print(standardIfElse(10));
19   }
20 }
```

String operators + and +=

- + and += can be used to concatenate strings
  - string concatenation
  - string conversion, if necessary
    - When the compiler sees a String followed by a '+' followed by a non-String, it attempts to convert the non-String into a String
- *Operator overloading*
  - Java programmers cannot implement their own overloaded operators like C++ and C# programmers can

String operators + and +=

Example

```
1  //: operators/StringOperators.java
2  import static net.mindview.util.Print.*;
3
4  public class StringOperators {
5    public static void main(String[] args) {
6      int x = 0, y = 1, z = 2;
7      String s = "x, y, z ";
8      print(s + x + y + z);
9      print(x + " " + s); // Converts x to a String
10     s += "(summed) = "; // Concatenation operator
11     print(s + (x + y + z));
12     print("" + x); // Shorthand for Integer.toString()
13   }
14 }
```

```
/* Output:
x, y, z 012
0 x, y, z
x, y, z (summed) = 3
0
*///:~
```

Operator Precedence

Operator	Description	Level	Associativity
[] . () ++ --	access array element access object member invoke a method post-increment post-decrement	1	left to right
++ -- + - ! ~	pre-increment pre-decrement unary plus unary minus logical NOT bitwise NOT	2	right to left
() new	cast object creation	3	right to left

Operator Precedence

* / %	multiplicative	4	left to right
+ - +	additive string concatenation	5	left to right
<< >> >>>	shift	6	left to right
< <= > >= instanceof	relational type comparison	7	left to right
== !=	equality	8	left to right
&	bitwise AND	9	left to right

Operator Precedence

^	bitwise XOR	10	left to right
|	bitwise OR	11	left to right
&&	conditional AND	12	left to right
||	conditional OR	13	left to right
?:	conditional	14	right to left
= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment	15	right to left

Thank you

zhenling@seu.edu.cn