Java : Data types, Identifiers and Operators

Instructor: Nihshanka Debroy

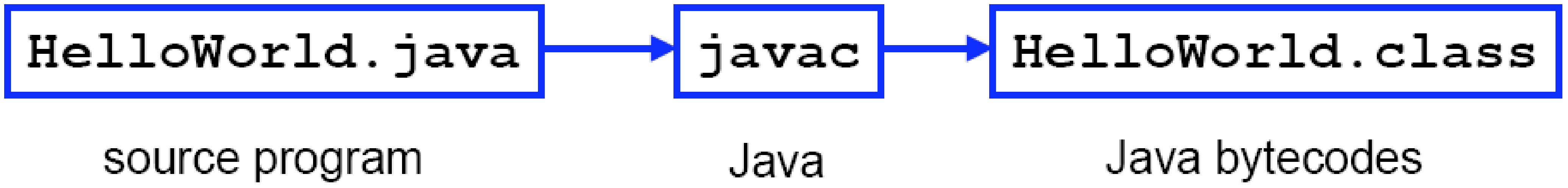
Java Programs

- Can use a text editor to type out Java code
- Save as a file with .java extension

  (Example: HelloWorld.java)
- File contains characters (stored as bytes)
- File cannot be directly executed by compiler -> translation into “bytecodes”
- Bytecode file executed by interpreter

Bytecodes

- Java *bytecode*
  - machine instruction for the Java processor
- Java compiler `javac` translates the source program into bytecodes
- Bytecode file has same name as the source program with a `.class` file extension: `HelloWorld.class`



Example from yesterday

```
public class Example1 {


/* @param args */


public static void
   main(String[] args)

{

System.out.println("This
   gets printed out.");

}

}
```

- Tells compiler you're creating a class called Example1 (so the java file should be Example1.java)
- Comments
- Function/method (set of statements grouped together), called main
- Beginning of main function
- Code statement: print a line of text, end with ;
- End of main function
- End of class

Memory

- Lowest level of abstraction: atoms
- Higher level: transistors (electronic switches)
- Everything represented as a collection of 0's and 1's

  (Yes/No, On/Off - Example: 10010010)
- Binary / Base-2 Number system (instead of decimal system)
- Bit (Binary Digit) -> single 0/1 entry in memory
- 8 bits = 1byte
- Larger measures
  - kilobyte (KB): 2^{10} = 1024 bytes
  - megabyte (MB): 2^{20} = 1,048,576 bytes
    - 1MB = 2^{10} KB
  - gigabyte (GB): 2^{30} = 1,073,741,824 bytes
    - 1GB = 2^{10} MB

A bit more about *bits*

- Compiler translates Java code to binary format
- Each character/number assigned a unique bit pattern
- Same set of 0's and 1's can represent different things – could denote a number, word, sentence, code, etc.
- Java handles memory management -> we only need to worry about data types

Definitions

- Variable: an item of data named by an identifier
- Operators:
  - Arithmetic
  - Relational and Conditional
  - Assignment
- Expression: “a series of variables, operators and method calls that evaluates to a single value”

Variables

- Think of them as labeled buckets that store information
- Can take the value out, put different values back in the bucket
- Similarly, computer is setting aside memory for our variable
- Name of a location in memory that holds a data value

Identifiers

- Each word in a computer program is an identifier -> 3 categories:

1) Identifiers that we choose: Example`1, args`

2) Identifiers that some other programmer chose: `String, System, out, println, main`

3) Identifiers that are reserved for special purposes in this programming language: `public, class, static, void`

```
public class Example1 {


/* @param args */


public static void main(String[]
   args)

{

System.out.println("This gets printed
   out.");

}

}
```

Naming Identifiers/Variable Declaration

- Any combination of letters, digits, underscore character (_) and dollar sign ($)
- Cannot begin with a digit
- Cannot be reserved word (int, for, while, etc.)
- Case sensitive (unique, UNIQUE, uniQUE different)
- Coding Conventions

What about ***myvar, name, 2cool, valid&ident*** ?

Data Types

- Computer memory stores arbitrary bit patterns
- Meaning of a bit pattern depends on its use
- Pattern used for a particular string of bits is a data type
- Categories:

  - Primitive (fundamental and built into Java)

  - Object (User-defined)

Primitive data types

- All primitive values belong to one of eight primitive types

  `byte   short   int   long   float`

  `double   char   boolean`

- Primitive data types use a fixed number of bytes
  - four of these types designate different sizes of bounded integers: `byte`, `short`, `int`, `long`
- A programmer can not create new primitive data types
- Any data type you invent will be a type of *object*
- Most commonly used types in practice: `int`, `boolean`, and `double`

Java primitive data types

Primitive Type	Description	Range
`byte`	8-bit integer	-128 to 127
`short`	16-bit integer	-32768 to 32767
`int`	32-bit integer	-2147483648 to 2147483647
`long`	64-bit integer	-2^{63} to $2^{63}-1$
`float`	32-bit floating point	10^{-46} to 10^{38}
`double`	64-bit floating point	10^{-324} to 10^{308}
`char`	Unicode character	
`boolean`	Boolean variable	`false` and `true`

More on Data Types

- Trade-off b/w memory used and what size value the data type can store
- Single bit: 2 values, 2 bits: 4 values, 3 bits: 8 values, and so on. N bits: 2^N values

  - byte uses 8 bits => 2^8= 256 values (-128 to 127)
- Signed: both +ve and -ve values
- Integers: values stored in binary notation
- Floating point numbers: bits divided to store sign, mantissa, and exponent

  Example: 2.99792458×10^8

Variable Declaration

Have to declare all variables before using them!

```
int number;
```

1) new variable of type “int”
2) having the name “number”

Examples

- `int x, y, z;`
- `int sum = 0;`
- `float f;`
- `double pi = 3.14;`

-
  ```
  char first  = 'T',
       middle = 'L',
       last   = 'B';
  ```

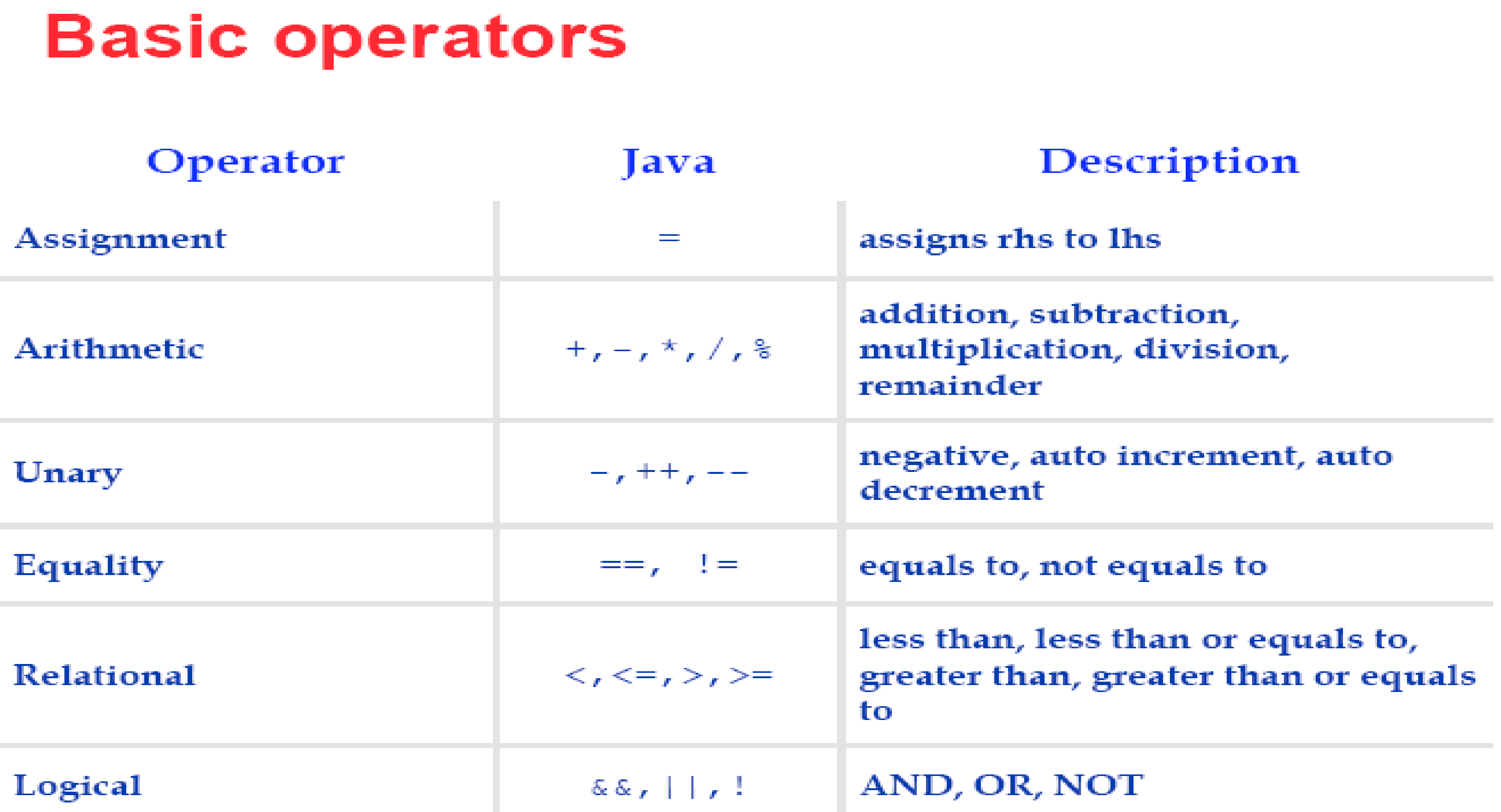
-
  ```
  char first  = 'T';
  char middle = 'L';
  char last   = 'B';
  ```

What's wrong in these ?

1) Int x;
2) float y
3) int float;
4) int 2good;
5) int yes&no;

Arithmetic Expressions

- Expressions: collections of operands (constants and variables) and operators
- Very similar to what you've seen in Math classes



Basic operators

Operator	Java	Description
Assignment	=	assigns rhs to lhs
Arithmetic	+, -, *, /, %	addition, subtraction, multiplication, division, remainder
Unary	-, ++, --	negative, auto increment, auto decrement
Equality	==, !=	equals to, not equals to
Relational	<, <=, >, >=	less than, less than or equals to, greater than, greater than or equals to
Logical	&&, ||, !	AND, OR, NOT

Examples

`int answer = 10 - 4;`

Division is different, depending on integer/floating point

- If both are integers (byte, short, int, long)=> integer division

  Example: `int answer = 5/2;` (remainders/fractions are dropped:answer will be 2)

- If one or both are floating point => floating point division

  Example: `double answer = 5/2.0;` (fraction parts saved: answer will be 2.5)

Remainder operator (mod operation): returns remainder

  Example: `int answer = 10%3;` (answer will be 1)

More Examples

1) X=2;

   X++;   (means X=X+1 –> so X will be 3)

2) a==b   (checks if a is equal to b)

3) a!=b   (checks if a not equal to b)

4) (a==b) &&(c==d) (checks if a = b and if c=d)

   (what if a=2, b=2, c=3, d=4 ?)

5) (a==b) || (c==d) (checks if a = b or if c=d)

   (what if a=2, b=2, c=3, d=4 ?)

6) if(!a)   (checks if a==0)

Operator precedence

- Evaluate `a + b * c`
  - multiplication first? `a + (b * c)`
  - addition first? `(a + b) * c`
- Java solves this problem by assigning priorities to operators (operator precedence)
  - operators with high priority are evaluated before operators with low priority
  - operators with equal priority are evaluated left to right

Operator priority (highest to lowest)

1. `( )`
2. `* / %`
3. `+ -`
4. `=`

When in doubt, use parentheses

- `a + b * c = a + (b * c)`
  - because * has higher priority than +
- To perform the + operation first we need to use parentheses
  - `(a + b) * c`
- If in any doubt use extra parentheses to ensure the correct order of evaluation
  - parentheses are free!
  - cause no extra work for the computer
  - only make it easier for you to work out what is happening

Examples

- Java adheres to traditional order of operations
- `*` and `/` have higher priority than `+` and `-`

```
int x = 3 + 5 * 6;           (x = 33)
int y = (3 + 5) * 6;         (y = 48)
```

- Parentheses are free, use them liberally

```
int z = ((3 + 5) * (6));     (z = 48)
```

- Equal priority operations are evaluated left-to-right in the absence of parentheses

```
int w = 3 * 4 / 2 * 6;             (w = 36)
int x = 3 * 4 / (2 * 6);     (x = 1)
int y = 3 * 4 + 2 * 6;             (y = 24)
int z = 3 * (4 + 2) * 6;     (z = 108)
```

Syntax and semantics

- **Addition, subtraction:** + **and** −, `int` **and** `double`

```
int x = 21+4;          (x = 25)
double y = 14.1-2;     (y = 12.1)
```

- **Multiplication:** *, `int` **and** `double`

```
int x = 21*4;          (x = 84)
double y = 14.1*2.5;   (y = 35.25)
```

- **Division:** /, **different for** `int` **and** `double`

```
int x = 21/4;          (x = 5)
double y = 21/4;       (y = 5.0)
double y = 21/4.0;     (y = 5.25)
```

- **Modulus:** %, **only for** `int`

```
int x = 21%4;          (x = 1)
```