COMP 110/L Lecture 9

Maryam Jalali

Slides adapted from Dr. Kyle Dewey

Outline

- Modulus (`%`) operator
- The `boolean` type
- `if` / `else`
  - Testing approaches with `if` / `else`

Modulus (%) Operator

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 5 / 2;
```

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 5 / 2;
```

```
x: 2
```

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 5 / 2;
```

`x:` 2 2 remainder 1

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 5 / 2;
```

`x:` 2 2 **remainder** 1

```
int x = 5 % 2;
```

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 5 / 2;
```

`x:` 2 — 2 **remainder** 1

```
int x = 5 % 2;
```

`x:` 1 — 2 **remainder** 1

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 1 / 2;
```

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 1 / 2;
```

```
x: 0
```

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 1 / 2;
```

`x:` `0` `0` **remainder** `1`

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 1 / 2;
```

`x:` `0` `0` **remainder** `1`

```
int x = 1 % 2;
```

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 1 / 2;
```

`x: 0` `0` **remainder** `1`

```
int x = 1 % 2;
```

`x: 1`

Modulus (%) Operator

Gets the remainder after division is performed on `int`s.

```
int x = 1 / 2;
```

`x: 0` `0` **remainder** `1`

```
int x = 1 % 2;
```

`x: 1` `0` **remainder** `1`

Example:

ModExample.java

Arithmetic Operators

Operator	meaning	Examples
+	plus - Add two operands	x+y
-	Minus - subtract right operand from the left	x-y
*	Multiplication- multiply two operands	x*y
/	Division - devide left operand by the right one	x/y
%	Modulus - remainder of the division of left operand by the right	x%y

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)
```

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 first perform the `+` operation 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 when the program is executing
  - 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)
```

boolean

`boolean`

- Represents the *truth value* of a question
- Only two possible values: `true` and `false`

`boolean`

- Represents the *truth value* of a question
- Only two possible values: `true` and `false`

```
boolean x = true;
```

boolean

- Represents the *truth value* of a question
- Only two possible values: `true` and `false`

```
boolean x = true;
```

```
boolean y = false;
```

Comparisons

`boolean` is useful for *arithmetic comparisons*

Comparisons

`boolean` is useful for *arithmetic comparisons*

```
boolean a = 5 > 1; // sets a to true
```

Comparisons

`boolean` is useful for *arithmetic comparisons*

```
boolean a = 5 > 1; // sets a to true
```

```
boolean b = 5 < 1; // sets b to false
```

Comparisons

`boolean` **is useful for *arithmetic comparisons***

```
boolean a = 5 > 1; // sets a to true
```

```
boolean b = 5 < 1; // sets b to false
```

```
boolean c = 5 <= 5; // sets c to true
```

Comparisons

`boolean` **is useful for *arithmetic comparisons***

```
boolean a = 5 > 1; // sets a to true
```

```
boolean b = 5 < 1; // sets b to false
```

```
boolean c = 5 <= 5; // sets c to true
```

```
boolean d = 6 >= 5; // sets d to true
```

Comparisons

`boolean` is useful for *arithmetic comparisons*

```
boolean e = 5 == 5; // sets e to true
```

Comparisons

`boolean` is useful for *arithmetic comparisons*

```
boolean e = 5 == 5; // sets e to true
```

```
boolean f = 5 == 6; // sets f to false
```

Comparisons

`boolean` is useful for *arithmetic comparisons*

```
boolean e = 5 == 5; // sets e to true
```

```
boolean f = 5 == 6; // sets f to false
```

```
boolean g = 5 != 5; // sets g to false
```

Comparisons

`boolean` is useful for *arithmetic comparisons*

```
boolean e = 5 == 5; // sets e to true
```

```
boolean f = 5 == 6; // sets f to false
```

```
boolean g = 5 != 5; // sets g to false
```

```
boolean h = 5 != 6; // sets h to true
```

Relational Operators – Relational operators are used to compare the value of operands (expressions) to produce a logical value. A logical value is either true or false.

Operators	Meaning	Example	Result
<	Less than	5>2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

String Concatentaion

Works as you might expect

String Concatentaion

Works as you might expect

```
true + "foo"
```

String Concatentaion

Works as you might expect

```
true + “foo”
“truefoo”
```

String Concatentaion

Works as you might expect

```
true + “foo”
```

```
“truefoo”
```

```
“bar” + false
```

String Concatentaion

Works as you might expect

```
true + “foo”
```

```
“truefoo”
```

```
“bar” + false
```

```
“barfalse”
```

Example:

Comparisons.java

if/else

`if` / `else`

Used to *conditionally* execute code
based on a `boolean` truth value

if / else

Used to *conditionally* execute code based on a `boolean` truth value

```
if (true) {
   System.out.println(“yes”);
} else {
   System.out.println(“no”);
}
```

if / else

Used to *conditionally* execute code based on a `boolean` truth value

```
if (true) {
   System.out.println("yes");
} else {
   System.out.println("no");
}
```

Prints `yes`

if / else

Used to *conditionally* execute code based on a `boolean` truth value

```
if (5 < 2) {
  System.out.println("yes");
} else {
  System.out.println("no");
}
```

if / else

Used to *conditionally* execute code based on a `boolean` truth value

```
if (5 < 2) {
  System.out.println("yes");
} else {
  System.out.println("no");
}
```

Prints `no`

Example:

IsGreaterThan5.java

Example:

MultipleReturn.java

Testing Advice with `if` / `else`

- Ideally, for each `if` / `else`, have *two* tests
  - One for if the condition is `true`
  - Another for if the condition is `false`

Testing Advice with `if` / `else`

- Ideally, for each `if` / `else`, have *two* tests
  - One for if the condition is `true`
  - Another for if the condition is `false`

Question: which tests may be good for testing absolute value?

Testing Advice with `if` / `else`

- Ideally, for each `if` / `else`, have *two* tests
  - One for if the condition is `true`
  - Another for if the condition is `false`

Question: which tests may be good for testing absolute value?

A positive value and a negative value

Example:

MultipleReturnTest.java