Numbers and Mathematics

Introduction to Programming and Computational Problem Solving II

CSE 8B

Lecture 3

Announcements

- Assignment 1 is due Jan 15, 11:59 PM
- Assignment 2 will be released Jan 15
  - Due Jan 23, 11:59 PM

Numbers and mathematics

- Numerical data types (e.g., an integer)
- Numeric operations (e.g., addition)
- Mathematical functions (e.g., cosine)
- Reading numbers from the console

Data types

- Java is a strongly typed language
  - **Programmers must explicitly identify the type of every variable, method, and object**

Numerical data types

Name	Range	Storage Size
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Number literals

- A literal is a constant value that appears directly in the program

```
int i = 34;
long x = 1000000;
double d = 5.0 + 1.0;
```

34, 100000, 5.0, and 1.0 are literals

Integer literals

- An integer literal can be assigned to an integer variable as long as it can fit into the variable
- A compilation error will occur if the literal is too large for the variable to hold
  - For example, the statement `byte b = 1000` would cause a compilation error, because `1000` cannot be stored in a variable of the `byte` type
- An integer literal is assumed to be of the `int` type, whose value is between -2^{31} (equals -2147483648) to $2^{31}-1$ (equals 2147483647)
- To denote an integer literal of the `long` type, append it with the letter `L` or `l`
  - `L` is preferred because `l` (lowercase `L`) can easily be confused with `1` (the digit one)

Floating-point literals

- Floating-point literals are written with a decimal point
- By default, a floating-point literal is treated as a `double` type value
  - Example: `5.0` is considered a `double` value, not a `float` value
- You can make a number a `float` by appending the letter `f` or `F`, and make a number a `double` by appending the letter `d` or `D`
  - Example: you can use `100.2f` or `100.2F` for a float number, and `100.2d` or `100.2D` for a double number

Scientific notation

- Floating-point literals can also be specified in scientific notation
  - Example: `1.23456e+2` (same as `1.23456e2`) is equivalent to `123.456`, and `1.23456e-2` is equivalent to `0.0123456`
- E or e represents an exponent

Numeric operations

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

`double` vs `float`

- The `double` type values are more accurate than the `float` type values
  - For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`

7 digits

Floating-point accuracy

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy
- For example,

  `System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);`

  displays `0.5000000000000001`, not `0.5`, and

  `System.out.println(1.0 - 0.9);`

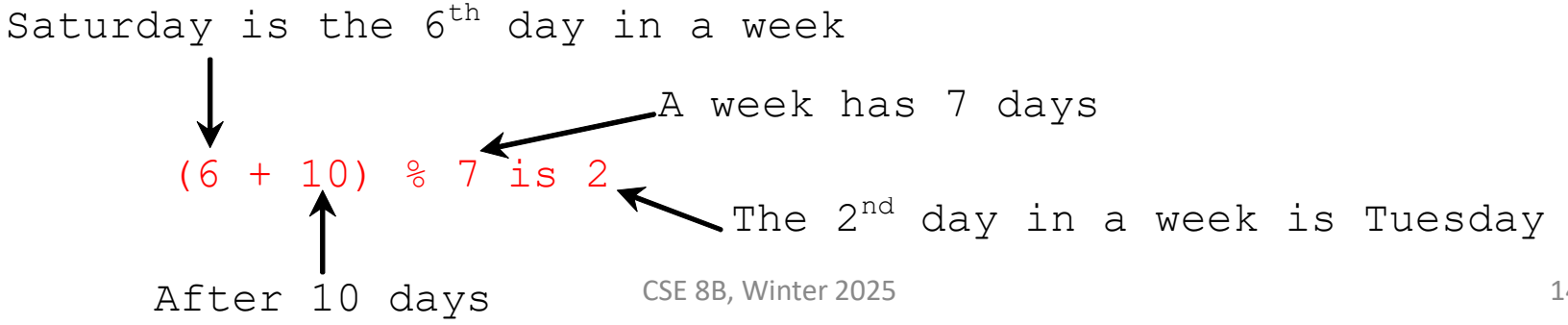
  displays `0.09999999999999998`, not `0.1`
- Integers are stored precisely
  - Calculations with integers yield a precise integer result

Integer division

- Warning: resulting fractional part (i.e., values after the decimal point) are **truncated, not rounded**
  - Example: 5 / 2 yields an integer 2

Remainder operator

- Example: an even number % 2 is always 0 and an odd number % 2 is always 1
  - You can use this property to determine whether a number is even or odd
- Example: If today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression.



Augmented assignment operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

Increment and decrement operators

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`

Conversion rules

- When performing a binary operation involving two operands of *different* types, Java automatically converts the operand based on the following rules
  1. If one of the operands is `double`, the other is converted into `double`
  2. Otherwise, if one of the operands is `float`, the other is converted into `float`
  3. Otherwise, if one of the operands is `long`, the other is converted into `long`
  4. Otherwise, both operands are converted into `int`

Type casting

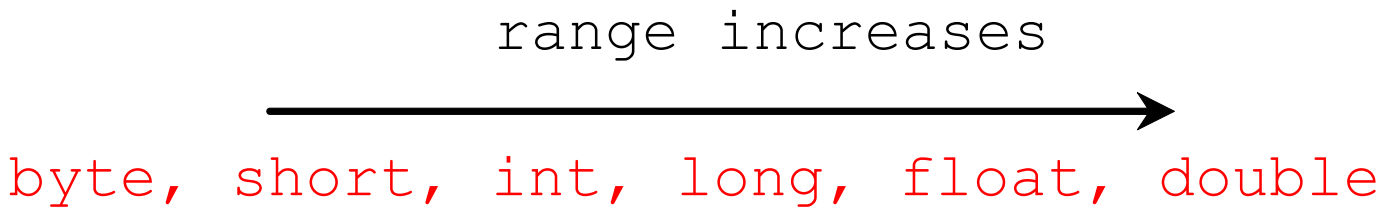
Implicit casting

```
double d = 3;
```
(type widening)

Explicit casting

```
int i = (int)3.0;
```
(type narrowing)

```
int i = (int)3.9;
```
(fraction part is truncated, not rounded!)



Operator precedence

- `()`, `var++`, `var--`
- `++var`, `--var`, `+`, `-` (unary plus and minus), `!` (not)
- (type) casting
- *, /, % (multiplication, division, and remainder)
- +, - (binary addition and subtraction)
- <, <=, >, >= (relational operators)
- ==, != (equality)
- ^ (exclusive or)
- && (and)
- || (or)
- =, +=, -=, *=, /=, %= (assignment operators)

Relational and logical operators will be covered next lecture

Operator associativity

- When two operators with the same precedence are evaluated, the associativity of the operators determines the order of evaluation
- All binary operators except assignment operators are left-associative

  `a - b + c - d` is equivalent to `((a - b) + c) - d`
- Assignment operators are right-associative

  `a = b += c = 5` is equivalent to `a = (b += (c = 5))`

Operator precedence and associativity

- The expression in the parentheses is evaluated first
  - Parentheses can be nested, in which case the expression in the inner parentheses is executed first
- When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule
- If operators with the same precedence are next to each other, their associativity determines the order of evaluation

Reading numbers from the console

- Create a `Scanner` object

```
Scanner input = new Scanner(System.in);
```

  - Java 8 API documentation
    - https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
  - Java 11 API documentation
    - https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html

Explicit import and implicit Import

- At top of source file

```
import java.util.Scanner; // Explicit Import

import java.util.*; // Implicit import
```

Reading numbers from the console

Method	Description
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

Reading numbers from the console

- Example: use the method `nextDouble()` to obtain to a `double` value

```
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

Mathematical functions

- Java provides many useful methods in the `Math` class for performing common mathematical functions
  - Java 8 API documentation
    - https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
  - Java 11 API documentation
    - https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Math.html

Mathematical functions

- Constants

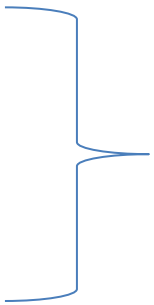
  `Math.PI`

  `Math.E`

- Math class methods
  - Trigonometric methods
  - Exponent methods
  - Rounding methods
  - `min`, `max`, `abs`, and `random` methods

Trigonometric methods

```
Math.toDegrees(radians)
Math.toRadians(degrees)
Math.sin(radians)
Math.cos(radians)
Math.tan(radians)
Math.acos(a)
Math.asin(a)
Math.atan(a)
```

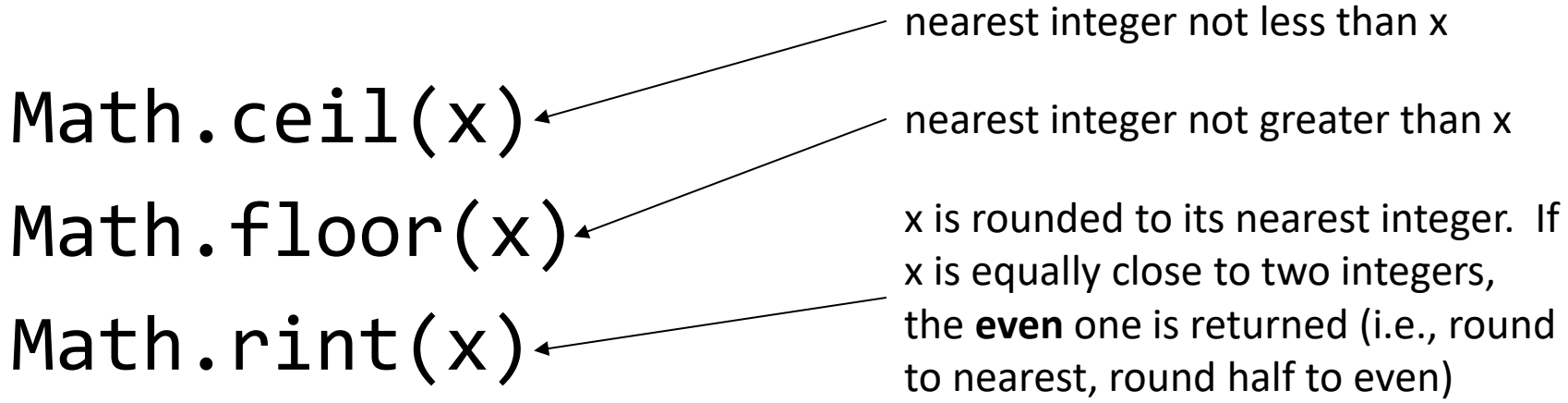


Result is in radians

Exponent methods

`Math.exp(a)`	e^a
`Math.log(a)`	$\log_e(a)$
`Math.log10(a)`	$\log_{10}(a)$
`Math.pow(a, b)`	a^b
`Math.sqrt(a)`	$\sqrt{a}$

Rounding methods



- If you want to return an integer type, then

  `int Math.round(float x)`

  - Returns `(int)Math.floor(x + 0.5f)`

  `long Math.round(double x)`

  - Returns `(long)Math.floor(x + 0.5)`

`min`, `max`, `abs`, and `random` methods

```
Math.min(a, b)
Math.max(a, b)
Math.abs(a)
Math.random()
```

- Returns a random `double` value in the range [0.0, 1.0)

Next Lecture

- Characters and strings