Chapter 2
Primitive Data

Data types

- **type**: A category or set of data values.
  - Constrains the operations that can be performed on data
  - Many languages ask the programmer to specify types

  - Examples: integer, real number, string

- Internally, computers store everything as 1s and 0s

```
104    → 01101000
"hi"   → 0110100 0110101
```

Java's primitive types

- **primitive types**: 8 simple types for numbers, text, etc.
  - Java also has **object types**, which we'll talk about later

Name	Description		Examples
`int`	integers	(up to $2^{31} - 1$)	`42, -3, 0, 926394`
`double`	real numbers	(up to 10^{308})	`3.1, -0.25, 9.4e3`
`char`	single text characters		`'a', 'X', '?', '\n'`
`boolean`	logical values		`true, false`

- Why does Java distinguish integers vs. real numbers?

Expressions

- **expression**: A value or operation that computes a value.
  - Examples:

```
1 + 4 * 5
(7 + 2) * 6 / 3
42
```

  - The simplest expression is a *literal value*.
  - A complex expression can use operators and parentheses.

Arithmetic operators

- **operator**: Combines multiple values or expressions.

`+`	addition
`-`	subtraction (or negation)
`*`	multiplication
`/`	division
`%`	modulus (a.k.a. remainder)

- As a program runs, its expressions are *evaluated*.
  - `1 + 1` evaluates to `2`
  - `System.out.println(3 * 4);` prints `12`
    - How would we print the text `3 * 4` ?

Integer division with /

- When we divide integers, the quotient is also an integer.
  - `14 / 4` is `3`, not `3.5`

```
   3
4 ) 14
   12
    2
```

```
   4
10 ) 45
   40
    5
```

```
   52
27 ) 1425
   135
    75
    54
    21
```

- More examples:
  - `32 / 5` is `6`
  - `84 / 10` is `8`
  - `156 / 100` is `1`

  - Dividing by 0 causes an error when your program runs.

Integer remainder with %

- The % operator computes the remainder from integer division.
  - `14 % 4` is `2`
  - `218 % 5` is `3`

```
      3
  4 ) 14
      12
      2
```

```
      43
  5 ) 218
      20
      18
      15
      3
```

What is the result?

```
45 % 6
2 % 2
8 % 20
11 % 0
```

- Applications of % operator:
  - Obtain last digit of a number: `230857 % 10` is `7`
  - Obtain last 4 digits: `658236489 % 10000` is `6489`
  - See whether a number is odd: `7 % 2` is `1`, `42 % 2` is `0`

Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.

    `1 - 2 - 3` is `(1 - 2) - 3` which is `-4`

  - But `* / %` have a higher level of precedence than `+ -`

    `1 + 3 * 4` is `13`

    ```
    6 + 8 / 2 * 3
    6 +   4   * 3
    6 +     12      is 18
    ```

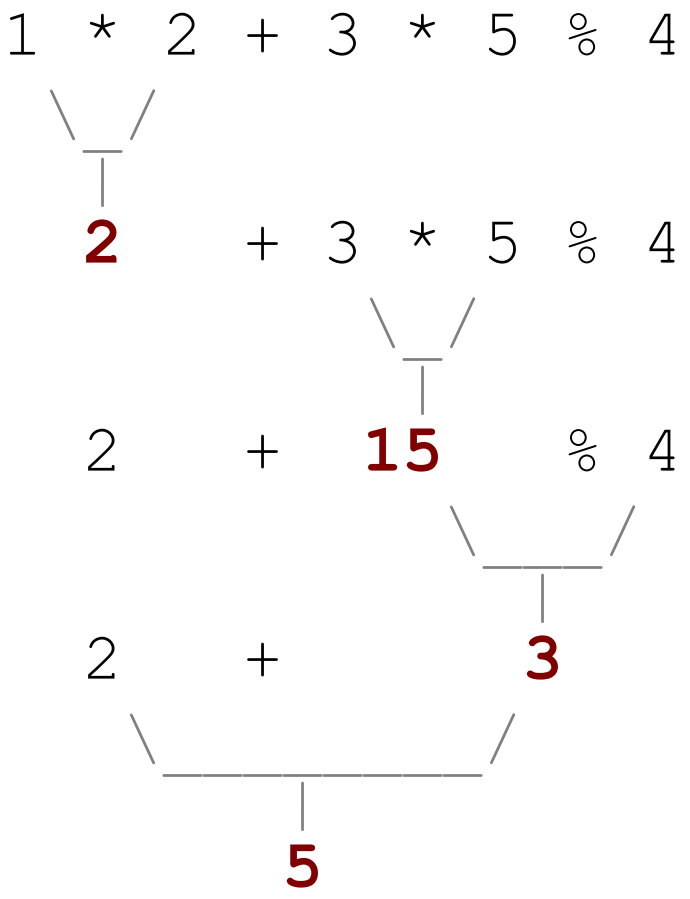
  - Parentheses can force a certain order of evaluation:

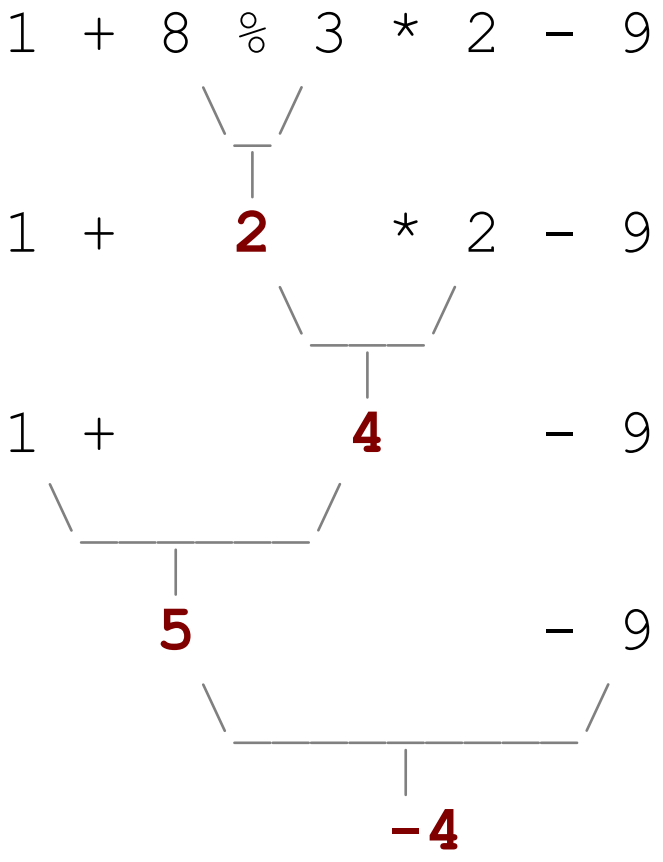
    `(1 + 3) * 4` is `16`

  - Spacing does not affect order of evaluation

    `1+3 * 4-2` is `11`

Precedence examples





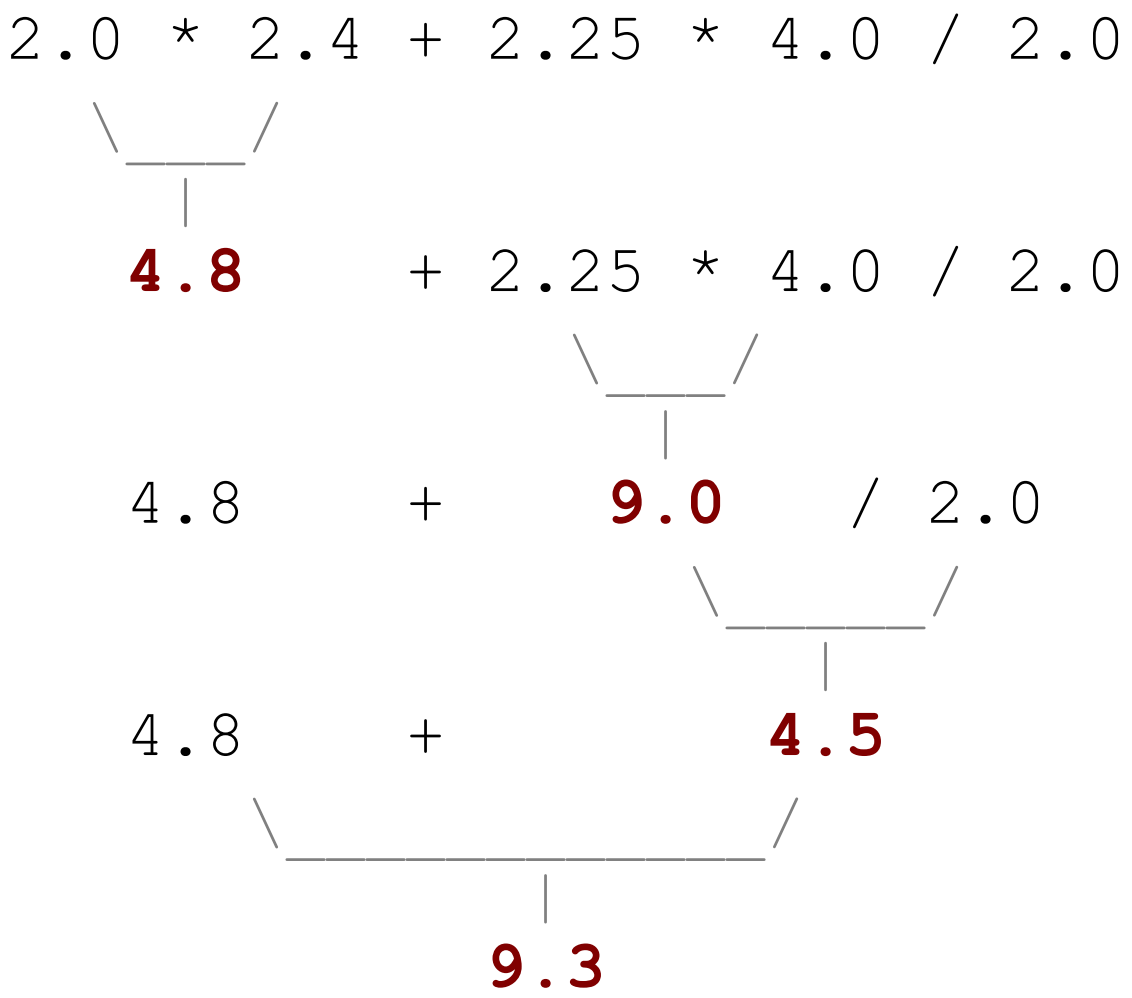
Precedence questions

- What values result from the following expressions?
  - `9 / 5`
  - `695 % 20`
  - `7 + 6 * 5`
  - `7 * 6 + 5`
  - `248 % 100 / 5`
  - `6 * 3 - 9 / 4`
  - `(5 - 7) * 4`
  - `6 + (18 % (17 - 12))`

Real numbers (type `double`)

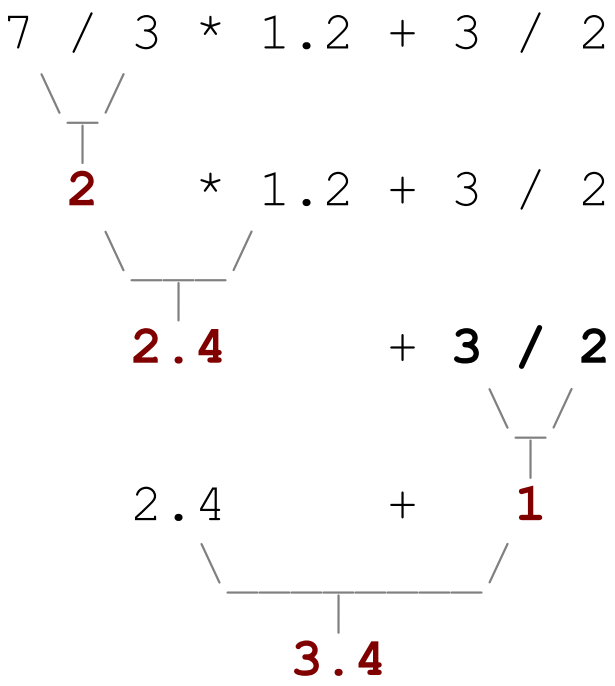
- Examples: `6.022`, `-42.0`, `2.143e17`
  - Placing `.0` or `.` after an integer makes it a `double`.
- The operators `+ - * / % ()` all still work with `double`.
  - `/` produces an exact answer: `15.0 / 2.0` is `7.5`
  - Precedence is the same: `()` before `* / %` before `+ -`

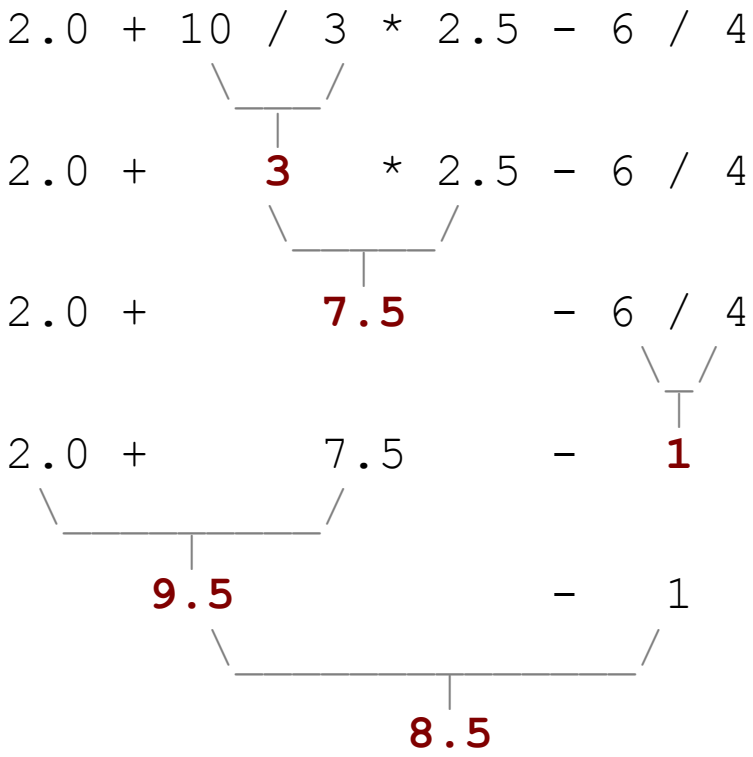
Real number example



Mixing types

- When `int` and `double` are mixed, the result is a `double`.
  - `4.2 * 3` is `12.6`
- The conversion is per-operator, affecting only its operands.





  - `3 / 2` is `1` above, not `1.5`.

String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

```
"hello" + 42   is "hello42"
1 + "abc" + 2  is "1abc2"
"abc" + 1 + 2  is "abc12"
1 + 2 + "abc"  is "3abc"
"abc" + 9 * 3  is "abc27"
"1" + 1        is "11"
4 - 1 + "abc"  is "3abc"
```

- Use + to print a string and an expression's value together.
  - `System.out.println("Grade: " + (95.1 + 71.9) / 2);`
    - Output: `Grade: 83.5`

Variables

Receipt example

What's bad about the following code?

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);
        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);
        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);
        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .08 +
                           (38 + 40 + 30) * .15);
    }
}
```

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `println` statements

Variables

- **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  - Like preset stations on a car stereo, or cell phone speed dial:

  - Steps for using a variable:
    - *Declare* it - state its name and type
    - *Initialize* it - store a value into it
    - *Use* it - print it or use it as part of an expression

Declaration

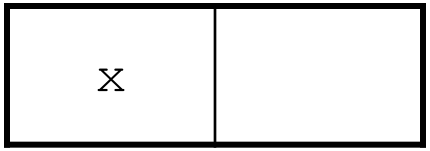
- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

- Syntax:

  type name;

    - The name is an *identifier*.

  - `int x;`



  - `double myGPA;`

Declaring Variables

```
int x;          // Declare x to be an
                // integer variable;

double radius; // Declare radius to
                // be a double variable;

char a;         // Declare a to be a
                // character variable;

int i,j, k;     //if variables are same kind
                //they can be declared in the same line
```

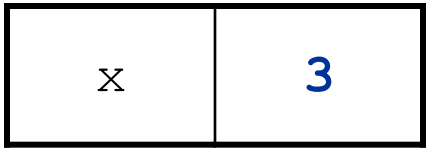
Assignment

- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- Syntax:

  name = **expression**;

  - `int x;`
    `x = 3;`



  - `double myGPA;`
    `myGPA = 1.0 + 2.25;`

Variable Assignment

```
int y = 1; // Assign 1 to variable y

double radius = 1.0; // Assign 1.0 to variable radius

int x = 5 * (3 / 2); // Assign the value of the expression to x

x = y + 1; // Assign the addition of y and 1 to x

double area = radius * radius * 3.14159; // Compute area
```

```
x = x + 1; //A variable can also be used in both sides of
the = operator.


1 = x; // Wrong
```

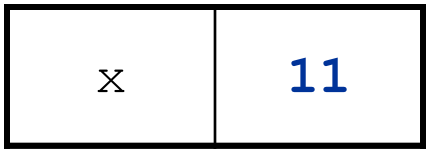
Using variables

- Once given a value, a variable can be used in expressions:

```
int x;
x = 3;
System.out.println("x is " + x);      // x is 3

System.out.println(5 * x - 1);        // 5 * 3 - 1
```

- You can assign a value more than once:




```
int x;
x = 3;
System.out.println(x + " here");      // 3 here

x = 4 + 7;
System.out.println("now x is " + x); // now x is 11
```

Declaration/initialization

- A variable can be declared/initialized in one statement.

- Syntax:

  type name = **value**;

  - `double myGPA = 3.95;`

myGPA	3.95

  - `int x = (11 % 3) + 12;`

x	14

Declaring and Initializing in One Step

- `int x = 1;`
- `double d = 1.4;`
- `int i= 3, j= 5;`

Assignment and algebra

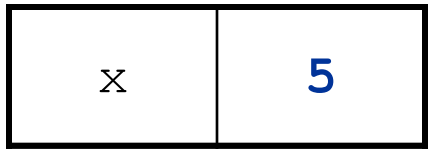
- Assignment uses = , but it is not an algebraic equation.

  = means, *"store the value at right in variable at left"*

  - The right side expression is evaluated first, and then its result is stored in the variable at left.

- What happens here?

```
int x = 3;
x = x + 2;    // ???
```



Assignment and types

- A variable can only store a value of its own type.
  - `int x = 2.5;` **`// ERROR: incompatible types`**

- An `int` value can be stored in a `double` variable.
  - The value is converted into the equivalent real number.
  - `double myGPA = 4;`

myGPA	4.0

  - `double avg = 11 / 2;`

avg	**5.0**

    - Why does `avg` store `5.0` and not `5.5` ?

Compiler errors

- A variable can't be used until it is assigned a value.
  -
```
int x;
System.out.println(x);    // ERROR: x has no value
```

- You may not declare the same variable twice.
  -
```
int x;
int x;                    // ERROR: x already exists
```

  -
```
int x = 3;
int x = 5;                // ERROR: x already exists
```

    - How can this code be fixed?

Printing a variable's value

- Use + to print a string and a variable's value on one line.

```
double grade = (95.1 + 71.9 + 82.6) / 3.0;
System.out.println("Your grade was " + grade);

int students = 11 + 17 + 4 + 19 + 14;
System.out.println("There are " + students +
                   " students in the course.");
```

- Output:

```
Your grade was 83.2
There are 65 students in the course.
```

Receipt question

Improve the receipt program using variables.

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.println("Subtotal:");
        System.out.println(38 + 40 + 30);

        System.out.println("Tax:");
        System.out.println((38 + 40 + 30) * .08);

        System.out.println("Tip:");
        System.out.println((38 + 40 + 30) * .15);

        System.out.println("Total:");
        System.out.println(38 + 40 + 30 +
                           (38 + 40 + 30) * .15 +
                           (38 + 40 + 30) * .08);
    }
}
```

Receipt answer

```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}
```

•If there is a number that does not require change, assign it to a **constant variable**.add keyword final to its declaration.

```
final datatype CONSTANTNAME = VALUE;
```

```
final double PI = 3.14159;
final int SIZE = 3;
```

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

Reading Input from the Console

1. Create a Scanner object

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

2. Use the method nextDouble() to obtain to a double value. For example,

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

```java
import java.util.Scanner;
public class SalaryCalculator {
    public static void main(String[] args) {
        int hourlyRate;
        Scanner scnr = new Scanner(System.in);
        System.out.println("Please enter the hourly rate as an integer:");
        hourlyRate = scnr.nextInt(); // this will convert the keyboard input to an integer.
        int annualSalary;
        annualSalary = hourlyRate * 40 * 50;
        System.out.println("The annual salary is:" + annualSalary + ".");
    }
}
```

- Getting input is achieved by first creating a Scanner object via the statement: `Scanner scnr = new Scanner(System.in);`
- System.in corresponds to keyboard input.
- Then, given Scanner object `scnr`, the following statement gets an input value and assigns x with that value: `x = scnr.nextInt();`.

Reading Numbers from the Keyboard

```
Scanner input = new Scanner(System.in);
int value = input.nextInt();
```

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.

Example:Compute Area With Console Input

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConsoleInput {
  public static void main(String[] args) {
    // Create a Scanner object
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter a radius
    System.out.print("Enter a number for radius: ");
    double radius = input.nextDouble();

    // Compute area
    double area = radius * radius * 3.14159;

    // Display result
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

Compute Average of 3 Numbers

```java
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAverage {
  public static void main(String[] args) {
    // Create a Scanner object
    Scanner input = new Scanner(System.in);

    // Prompt the user to enter three numbers
    System.out.print("Enter three numbers: ");
    double number1 = input.nextDouble();
    double number2 = input.nextDouble();
    double number3 = input.nextDouble();

    // Compute average
    double average = (number1 + number2 + number3) / 3;

    // Display result
    System.out.println("The average of " + number1 + " " + number2
      + " " + number3 + " is " + average);
  }
}
```

Numeric Operators

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

Integer Division

+, -, *, /, and %

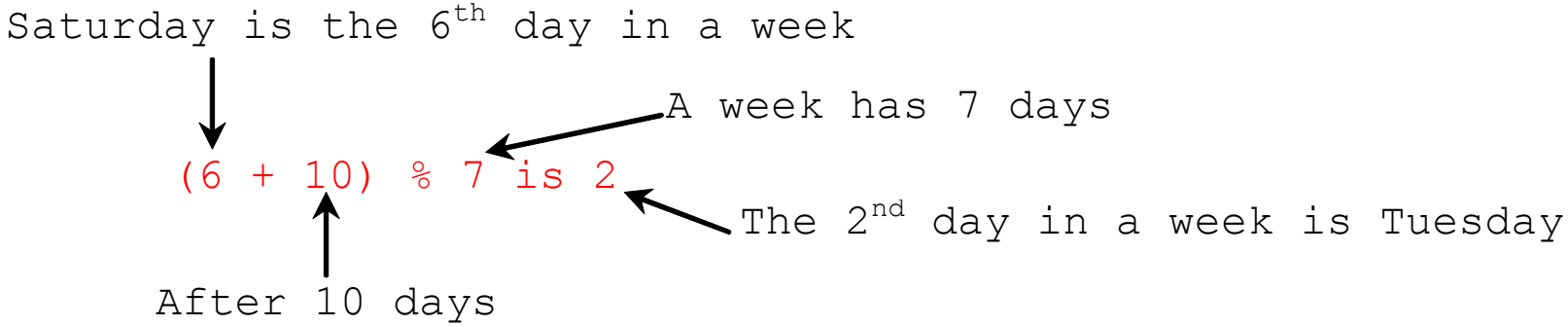
5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)

Remainder Operator

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd. Suppose 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:



Math Methods

A **method** is a list of statements executed by invoking the method's name. A **method call** invokes the execution of this list automatically. For example, `Math.sqrt(3)` invokes the statements that calculate the square root of 3. The value(s) inside `()` is called the **argument(s)**.

- `Math.sqrt()` : returns the square root of the argument.
- `Math.abs()` : returns the absolute value of the argument.
- `Math.pow(x, y)` : returns the y-th power of x.
- `Math.random()` : returns a random number between 0.0 and 0.9999....
- Activity 2.10.6: Method calls in arguments.
- Challenge 2.10.3: Using math functions to calculate the distance between two points.

```
xDist = x2 - x1;
yDist = y2 - y1;
pointsDistance = Math.sqrt(Math.pow(xDist, 2) + Math.pow(yDist, 2));
```

Exponent Operations

```
System.out.println(Math.pow(2, 3));
// Displays 8.0
System.out.println(Math.pow(4, 0.5));
// Displays 2.0
System.out.println(Math.pow(2.5, 2));
// Displays 6.25
System.out.println(Math.pow(2.5, -2));
// Displays 0.16
```

Number Literals

A *literal* is a constant value that appears directly in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

```
int i = 34;

long x = 1000000;

double d = 5.0;
```

Integer Literals

An integer literal can be assigned to an integer variable as long as it can fit into the variable. A compilation error would occur if the literal were 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} (-2147483648) to $2^{31}-1$ (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).

Scientific Notation

Floating-point literals can also be specified in scientific notation, for 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 and it can be either in lowercase or uppercase.

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`

Type Conversions

- **Implicit Conversions**: if a type conversion is needed and can be done without loss of precision, the compiler will perform it automatically.
  - For an arithmetic operator like + or *, if either operand is a double, the other is automatically converted to double, and then a floating-point operation is performed.
  - For assignments, the right side type is converted to the left side type if the conversion is possible without loss of precision.
- **Type casting**: convert an item's type explicitly.
  - Format: (type)variable.
  - Example: `myInt = (int)myDouble;`
  - Exercise: Family 1 has 3 children. Family 2 has 4 children. Family 3 has 1 child. Write a program that calculates the average number of childern per family.
  - Common error: accidentally perform integer division when floating-point division was intended.
  - Activity 2.12.5: Type casting.

```
avgKids = (double)(numKidsA + numKidsB + numKidsC) / numFamilies;
```

Characters

The `char` type variable is used to store a single character. In java, a character value should be surrounded by single quotes, as in `myChar = 'm';`.

- A character is internally stored as a number
  - **ASCII** is an early standard for encoding characters as numbers.
  - Table 2.14.1: ASCII table
  - Activity 2.14.2: A char variable stores a number.
  - Character arithmetic
- Escape sequence
  - In addition to regular characters, character encoding includes several special characters, such as newline or tab.
  - The special characters are represented as **escape sequences**, which consist of a backslash `\` and another character.
  - Table 2.14.2: Common escape sequences.
  - Activity 2.14.5: Escape sequences.
- Getting a character from input
  - Java scanner does not have a method for getting one character from input.
  - Instead, use the following sequence: `myChar = scnr.next().charAt(0);`

Strings

A **string** is a sequence of characters. In java, a string is represented using text surrounded by double quotes, as in `"Hello"`.

- Activity 2.15.1: A string is stored as a sequence of characters in memory.
- String variables and assignments.
  - Use type `String`. Note that the word is capitalized, becasue its underlying structure is different from a primitive type such as int or double.
- Get a string from input
  - Load a whole line of text: `scnr.nextLine()`
  - Load text until the next whitespace: `scnr.next()`
  - Mixing `next()` and `nextLine()` can be tricky because of the invisible newline character.
  - Activity 2.15.7: Combining next() and nextLine() can be tricky.
  - Challenge 2.15.1: Reading and outputting strings.
- String methods