BIRZEIT UNIVERSITY

Methods

By: Mamoun Nawahdah (Ph.D.)
2022

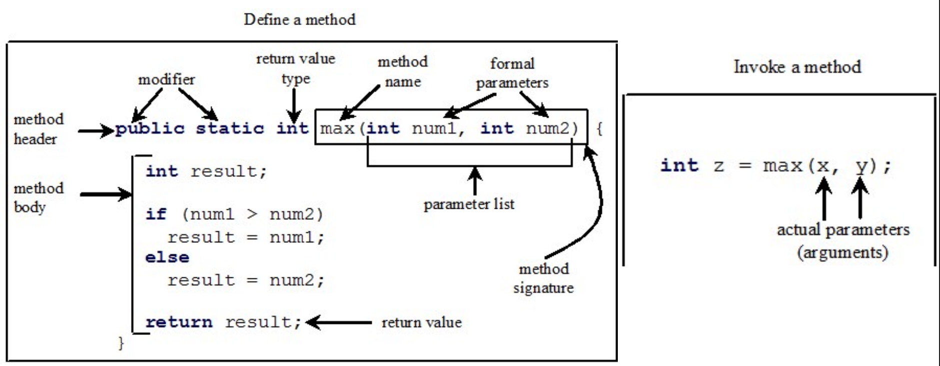
Opening Problem

Find the sum of integers from 1 to 10, from 20 to 30, and from 35 to 45, respectively.

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
  sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
  sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

Defining Methods

❖ A method is a collection of statements that are grouped together to perform an operation.



3

Defining Methods

❖ *Method signature* is the combination of the method name and the parameter list.

❖ The variables defined in the method header are known as *formal parameters*.

❖ When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.

❖ A method may return a value. The returnValueType is the data type of the value the method returns. If the method does not return a value, the returnValueType is the keyword void. For example, the returnValueType in the main method is void.

CAUTION

- A **return** statement is required for a value-returning method.
- The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it possible that this method does not return any value.

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else if (n < 0)
    return -1;
}
```

(a)

Should be

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else
    return -1;
}
```

(b)

- To fix this problem, delete ***if (n < 0)*** in (a), so that the compiler will see a **return** statement to be reached regardless of how the **if** statement is evaluated.

5

Passing Parameters

```
public static void nPrintln(String message, int n) {
    for (int i = 0; i < n; i++)
        System.out.println(message);
}
```

- Suppose you invoke the method using
  nPrintln("Welcome to Java", 5);
  What is the output?
- Suppose you invoke the method using
  nPrintln("Computer Science", 15);
  What is the output?
- Can you invoke the method using
  nPrintln(15, "Computer Science");

6

Case Study: Converting Hexadecimals to Decimals

Write a method that converts a hexadecimal number into a decimal number.

ABCD =>

A*16^3 + B*16^2 + C*16^1+ D*16^0

= ((A*16 + B)*16 + C)*16+D

= ((10*16 + 11)*16 + 12)*16+13 = ?

Ambiguous Invocation

Overloading

```
public class Test {
    public static void main(String[] args) {
        System.out.println(max(1, 2));
    }

    public static double max(int num1, double num2) {
        if (num1 > num2)
          return num1;
        else
          return num2;
      }

      public static double max(double num1, int num2) {
        if (num1 > num2)
          return num1;
        else
          return num2;
      }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
	The method max(int, double) is ambiguous for the type Test

	at Test.main(Test.java:3)
```

Scope of Local Variables

- A **local variable**: a variable defined inside a **block** (e.g. method, loop).
- **Scope**: the part of the program where the variable can be referenced.
- The scope of a local variable **starts from its declaration and continues to the end of the block that contains the variable**.
- A local variable **must** be declared before it can be used.

9

Scope of Local Variables

- You can declare a local variable with the same name multiple times in different **non-nesting** blocks in a method, but you cannot declare a local variable twice in nested blocks.

It is fine to declare i in two non-nesting blocks

```
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x += i;
  }

  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

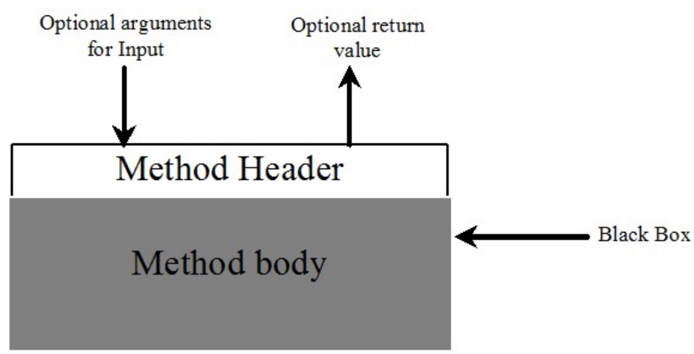
It is wrong to declare i in two nesting blocks

```
public static void method2() {

  int i = 1;
  int sum = 0;

  for (int i = 1; i < 10; i++)
    sum += i;
  }

}
```

10

Method Abstraction

❖ You can think of the method body as a black box that contains the detailed implementation for the method.



11

Benefits of Methods

- Write a method once and **reuse** it anywhere.
- **Information hiding**. Hide the implementation from the user.
- **Reduce complexity**.

12

The Math Class

- Class constants:
  - **PI**
  - **E**
- Class methods:
  - Trigonometric Methods
  - Exponent Methods
  - Rounding Methods
  - min, max, abs, and random Methods

Trigonometric Methods

- **sin(double a)**
- **cos(double a)**
- **tan(double a)**
- **acos(double a)**
- **asin(double a)**
- **atan(double a)**

Radians

Math.toRadians(90)

Examples:

Math.sin(0)	returns 0.0
Math.sin(Math.PI / 6)	returns 0.5
Math.sin(Math.PI / 2)	returns 1.0
Math.cos(0)	returns 1.0
Math.cos(Math.PI / 6)	returns 0.866
Math.cos(Math.PI / 2)	returns 0.0

Exponent Methods

- **exp(double a)**
  Returns e raised to the power of a.
- **log(double a)**
  Returns the natural logarithm of a.
- **log10(double a)**
  Returns the 10-based logarithm of a.
- **pow(double a, double b)**
  Returns a raised to the power of b.
- **sqrt(double a)**
  Returns the square root of a.

Examples:

Math.exp(1)	returns 2.71
Math.log(2.71)	returns 1.0
Math.pow(2, 3)	returns 8.0
Math.pow(3, 2)	returns 9.0
Math.pow(3.5, 2.5)	returns 22.917
Math.sqrt(4)	returns 2.0
Math.sqrt(10.5)	returns 3.24

Rounding Methods

- **double ceil(double x)** x rounded up to its nearest integer. This integer is returned as a double value.
- **double floor(double x)** x is rounded down to its nearest integer. This integer is returned as a double value.
- **double rint(double x)** x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double.
- **int round(float x)** Return (int)Math.floor(x+0.5).
- **long round(double x)** Return (long)Math.floor(x+0.5).

min, max, and abs

- max(a, b) and min(a, b)

  Returns the maximum or minimum of two parameters.

- abs(a)

  Returns the absolute value of the parameter.

- random()

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

Examples:

Math.max(2, 3)	returns 3
Math.max(2.5, 3)	returns 3.0
Math.min(2.5, 3.6)	returns 2.5
Math.abs(-2)	returns 2
Math.abs(-2.1)	returns 2.1

17

The random Method

- Generates a random double value greater than or equal to 0.0 and less than 1.0

(0 <= Math.random() < 1.0)

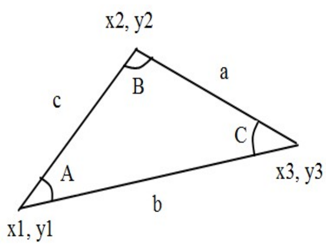
```
(int)(Math.random() * 10)  →  Returns a random integer between 0 and 9.

50 + (int)(Math.random() * 50)  →  Returns a random integer between 50 and 99.
```

In general:

```
a + Math.random() * b  →  Returns a random number between a and a + b, excluding a + b.
```

18

Case Study: Computing Angles of a Triangle




```
A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))
```

Write a program that prompts the user to enter the x- and y-coordinates of the three corner points in a triangle and then displays the triangle's angles.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

19

Problem: Guessing Numbers

- ❖ Write a program that generate a random integer between 0 and 100, inclusive.
- ❖ The program prompts the user to enter a number continuously until the number matches the random number.
- ❖ For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently.