

Lecture 64

Java – Methods [Static Method]

Mostafa Elhosseini

Professor at College of Computer Science and
Engineering

Taibah University

<https://youtube.com/drmelhosseini>

| Program Units in java

- Java **Application Programming Interface** (also referred to as the Java **API** or **Java class library**)
- **Related classes** are typically grouped into **packages** so that they can be imported into programs and reused
- Java 9 introduces another program unit called **modules**

| Program Units in java

- The **Java API** provides a rich collection of predefined classes that contain methods for performing common **mathematical calculations, string manipulations, character manipulations, input/output operations, database operations, networking operations, file processing, error checking and more**

| Why Methods?

- **Software reusability**—using existing classes and methods as building blocks to create new programs.
- Methods help you **modularize a program** by separating its tasks into self-contained units
- A third motivation is to **avoid repeating code**.
- Dividing a program into meaningful methods makes the program **easier to debug and maintain**

| Static Method, and Static Variables

- Sometimes a **method** performs a task that **does not depend on an object**.
- Such a method applies to the class in which it's **declared** as a whole and is known as a **static method** or a **class method**
 - Math's static method **pow** to raise a value to a power
- For any **class** imported into your program, you can call the **class's static methods** by specifying the **class's name, followed by a dot (.)** and the **method name**

| Math class Methods

- Class **Math** provides a collection of methods that enable you to perform common mathematical calculations
- **Math.sqrt(900.0)**
- Method **sqrt** takes an argument of **type** double and **returns** a result of type **double**
- **System.out.println(Math.sqrt(900.0));**
- There was **no need to create a Math object** before calling method **sqrt**
- **All Math class** methods are **static**—therefore, each is called by preceding its name with the class name **Math** and the dot (.) separator

| Math class Methods

- Class **Math** is part of the **java.lang** package, which is **implicitly imported by the compiler**, so it's **not necessary to import** class **Math** to use its methods
- **Method arguments** may be **constants, variables or expressions**.
- If **c = 13.0**, **d = 3.0** and **f = 4.0**, then the statement **System.out.println(Math.sqrt(c + d * f));**

| Math class Methods

Method	Description	Example
<code>abs(x)</code>	absolute value of x	<code>abs(23.7)</code> is 23.7 <code>abs(0.0)</code> is 0.0 <code>abs(-23.7)</code> is 23.7
<code>ceil(x)</code>	rounds x to the smallest integer not less than x	<code>ceil(9.2)</code> is 10.0 <code>ceil(-9.8)</code> is -9.0
<code>cos(x)</code>	trigonometric cosine of x (x in radians)	<code>cos(0.0)</code> is 1.0
<code>exp(x)</code>	exponential method e^x	<code>exp(1.0)</code> is 2.71828 <code>exp(2.0)</code> is 7.38906
<code>floor(x)</code>	rounds x to the largest integer not greater than x	<code>floor(9.2)</code> is 9.0 <code>floor(-9.8)</code> is -10.0
<code>log(x)</code>	natural logarithm of x (base e)	<code>log(Math.E)</code> is 1.0 <code>log(Math.E * Math.E)</code> is 2.0
<code>max(x, y)</code>	larger value of x and y	<code>max(2.3, 12.7)</code> is 12.7 <code>max(-2.3, -12.7)</code> is -2.3
<code>min(x, y)</code>	smaller value of x and y	<code>min(2.3, 12.7)</code> is 2.3 <code>min(-2.3, -12.7)</code> is -12.7
<code>pow(x, y)</code>	x raised to the power y (i.e., x^y)	<code>pow(2.0, 7.0)</code> is 128.0 <code>pow(9.0, 0.5)</code> is 3.0
<code>sin(x)</code>	trigonometric sine of x (x in radians)	<code>sin(0.0)</code> is 0.0
<code>sqrt(x)</code>	square root of x	<code>sqrt(900.0)</code> is 30.0
<code>tan(x)</code>	trigonometric tangent of x (x in radians)	<code>tan(0.0)</code> is 0.0

| Class Variables

- Just as a class may contain static methods, a **class** also may contain **static variables**.
- Such variables are also known as **class variables**.
- Class Math declares two **constants**, **Math.PI** and **Math.E**

| Class Variables

These **constants** are **declared** in class **Math** with the **modifiers public, final and static**:

- Making them **public** allows you **to use them in your own classes**.
- Any variable declared with keyword **final** is **constant**—its value cannot change after the variable is initialized.
- Making these variables **static** allows them to be **accessed via the class name** **Math** and a dot (.) separator, just as class **Math**'s methods are.
- A **class's variables** are sometimes called **fields**

| Note:

- when you invoke a Method, java remembers the place where it happened and jumps into the invoked method;
- the body of the method is then executed;
- reaching the end of the method forces java to return to the place directly after the point of invocation.

| Function arguments

- Mathematical Methods usually take one argument, e.g., $\sin(x)$ takes an x , which is the measure of an angle.
 - May accept any number of arguments
 - Any number includes zero - some java Methods don't need any argument
 - java Methods strongly demand the presence of a pair of parentheses - opening and closing ones, respectively
 - If you're going to use a Method which doesn't take any argument, you still have to have the parentheses.
 - to distinguish ordinary words from Methods names, place a pair of empty parentheses after their names

| What happens when java encounters an invocation?

- First, java checks if the name specified is legal (it browses its internal data in order to find an existing Method of the name)
- Second, java checks if the Method's requirements for the number of arguments allows you to invoke the method in this way
- Third, java leaves your code for a moment and jumps into the method you want to invoke; fourth, the method executes its code
- Finally, java returns to your code

| References

- Java: How to Program, 9th Edition (Deitel) 9th Edition by Paul Deitel , Harvey M. Deitel
- Head First Java, 2nd Edition 2nd Edition by Kathy Sierra, Bert Bates
- Think Java: How to Think Like a Computer Scientist 1st Edition by Allen B. Downey, Chris Mayfield