

JAVA / LESSON 01 OF 40

Steve's First Line

Start with one line of output and build your first code house

Java 11+ | 90 minutes



Create a server welcome screen in the console

The program will not guess what you mean. If you write "welcome", it will say welcome; if you write one less symbol, it may not even be able to say the first sentence. In this lesson, we start from a complete program, learn to run it, control line breaks, understand error reports, and finally create a server welcome screen with our own style.

Before you start: No programming experience required

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=1&lang=en>

Long code lines may wrap at the > margin marker. The marker is not part of the code.

In this lesson

01. How can a computer speak?
02. The first complete program: find main
03. From source file to running result
04. print and println: Who is responsible for line breaks?
05. Small symbols are not decoration
06. Comments: Notes for Your Future Self
07. First debugging: Let differences become clues
08. Mini-project: A Server Welcome Screen
09. Check Your Work: Is the Welcome Screen Clear?
10. Take another small step forward

01 / 10

How can a computer speak?



The workbench in the illustration provides the background to the story; the real output appears in the development tool's console.

Steve stood in front of the workbench and wanted to tell his partner: "Bring a torch first, and then enter the mine." However, the computer did not know which sentence was important. It requires a clearly written instruction. This is the most interesting part of programming: turning a vague idea into steps that others can execute.

Read on with three questions

- Where does the program start executing?
- Why should text be placed in quotation marks?
- After finishing a sentence, how to decide whether to change the line?

First prepare a Java development tool with JDK installed, using Java 11 or newer version. Just create a normal Java project; these lessons do not require connecting to Minecraft or installing mods.

02 / 10

The first complete program: find main

Create new HelloWorld.java. The file name must be consistent with the name after the public class, even in upper and lower case. If the tool has already generated main, use the existing entry instead of adding another layer of templates.

Complete program · HelloWorld.java

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, block world!");  
    }  
}
```

Output

```
Hello, block world!
```

Read the code, first look at the structure

Location	Effect
class HelloWorld	Name the classes in this code
main(...)	Execution entry of this lesson program
System.out.println(...)	Send text to console
Two pairs { }	The outer layer is the class, and the inner layer is the scope of main

Don't rush to memorize public, static and void. Recognizing the entrance, finding the two pairs of braces, and making the first sentence appear successfully are the goals of this page.

From source file to running result



You are responsible for expressing the idea; the compiler and JVM are responsible for getting the rules onto the machine.

A four-stop journey of a piece of code

Step	Explanation
1. .java	Write the source code
2. javac	Check syntax and compile
3. .class	Get bytecode
4. JVM	Execute the program and output

After saving the file, the development tool will usually compile and run it for you. Compilation first checks whether the code complies with Java rules; at runtime, the JVM executes the generated bytecode. The program may still encounter problems while running, so "compiling and passing" is only the first hurdle.

Want to see what the tool does for you?

Optional: terminal commands to run in the source file directory

```
javac -encoding UTF-8 HelloWorld.java
java HelloWorld
```

The first line compiles the source file; the second line runs the class without adding .java or .class suffix. If the project automatically adds a package statement, priority is given to using the run button of the development tool. The terminal example in this lesson is written as a file without package.

04 / 10

print and println: Who is responsible for line breaks?

Think of the console as a sticky note that keeps writing down. After print finishes writing the content, it stops where it is; println finishes writing the content and then sends the next output to a new line. Spaces within quotation marks will also appear unchanged.

One line to continue, or the next line?

Element	Explanation
print	Write content Stop on the same line
println	Write content Line break at the end
Next output	Starting from the current cursor position

Snippet: place inside main and run separately

```
System.out.print("diamond");  
System.out.print(" + ");  
System.out.println("stick");  
System.out.println("The first layer: grass block");  
System.out.println("Second layer: soil");  
System.out.println();  
System.out.print("Hiss..");  
System.out.println("Run quickly!");
```

Output

```
diamond + stick  
The first layer: grass block  
Second layer: soil  
  
Hiss..Run quickly!
```

Println() without content will also wrap, so a blank line appears here. First draw the text of each line, and then run the verification; "one blank line missing" is also an interpretable experimental result.

05 / 10

Small symbols are not decoration



An inconspicuous missing connection can bring an entire redstone line to a halt.

The symbols in the code have their own responsibilities. A semicolon ends the declaration or output statement of this lesson; curly braces delimit the scope; parentheses contain the content required by the method; English double quotes mark the text; double slashes start a comment.

Three snippets with mistakes

Deliberately written wrong: check separately, do not copy and run the entire section

```
System.out.println("The creeper is coming")
System.out.println("Run quickly!");
System.out.Println("Welcome!");
```

- The first line is missing a semicolon, fill in ;.
- The second line used Chinese curly quotation marks and changed them to English double quotation marks ".
- The case of Println in the third line is wrong, it should be println.

The line number pointed out by the compiler is sometimes the location where it "finally realizes something is wrong." Check that line and see if the previous line is missing quotes, semicolons, or parentheses.

06 / 10

Comments: Notes for Your Future Self



When troubleshooting, a clear clue is more useful than changing things everywhere.

Good comments explain the reasons behind the choices. Half a year later, you will often need a little background just like your friends who read this code for the first time. Comments do not become console output, nor do they perform tasks for the program.

Snippet: place inside main and run separately

```
// First display the safety prompt, and then introduce today's task.
System.out.println("Bring a torch to avoid getting lost.");

/* The welcome screen is only responsible for displaying text.
   Input and command processing will be studied in later courses. */
System.out.println("Today's task: collect logs.");
```

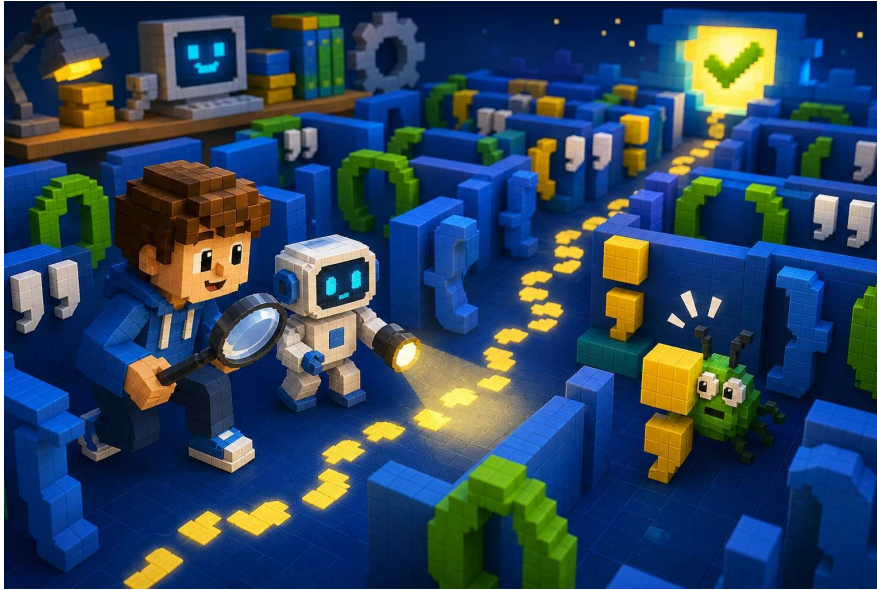
Output

```
Bring a torch to avoid getting lost.
Today's task: collect logs.
```

A comment that merely says 'print a welcome message' adds little to a print statement. Explain why the code is arranged this way or what its limitations are.

07 / 10

First debugging: Let differences become clues



Debugging is not about arguing with the computer, but about finding where "expected" and "actual" separate.

When you see the red line, stop for a moment. Find the first error message related to your file, read the file name and line number, and then check for nearby symbols. Only change one reason at a time, save and recompile. In this way, you will know which step worked.

Error finding experiment: This program cannot be compiled on purpose

```
public class BugHunt {  
    public static void main(String[] args) {  
        System.out.println("I will find it!")  
    }  
}
```

Answer first, then take action

- What's missing at the end of the output statement?
- Is it a compilation error or an error after running?
- Which line should be output after repair?

Answer: Add the missing semicolon. This is a compile-time error. The corrected program prints 'I will find it!'. If a program runs but displays the wrong message, compare the content and order of its output.

08 / 10

Mini-project: A Server Welcome Screen

A good welcome screen will tell readers: what is this place called, who is here, what to do today, and what should be paid attention to. First arrange the lines on the paper, and then let each line correspond to an output statement.

Split the welcome screen into four tasks

Step	Explanation
1. Border	Beginning and end
2. Identity	Server and players
3. Task	Today's goal
4. Hint	Safety and fun

Complete program · ServerWelcome.java

```
public class ServerWelcome {  
    public static void main(String[] args) {  
        System.out.println("=====");  
        System.out.println("Welcome to: Cube Adventure Camp");  
        System.out.print("Player: ");  
        System.out.println("Alex");  
        System.out.println("Today's task: collect 10 Logs");  
        // Safety tips are placed in front of interesting tips.  
        System.out.println("Safety Tip: Do not dig straight down");  
        System.out.println("Fun tip: Creepers want to join the lesson too. Keep your distance!");  
        System.out.println("=====");  
    }  
}
```

This is the welcome screen in the console. It doesn't yet send text to the game server; however, the formatting, task breakdown, and troubleshooting methods can already be carried over to the next program.

09 / 10

Check Your Work: Is the Welcome Screen Clear?

Complete output of the reference program on the previous page

```
=====
Welcome to: Cube Adventure Camp
Player: Alex
Today's task: collect 10 Logs
Safety Tip: Do not dig straight down
Fun tip: Creepers want to join the lesson too. Keep your distance!
=====
```

The ability map of the first lesson

Step	Explanation
1. Find the entrance	main
2. Control output	print / println
3. Check mark	Quotes and semicolons
4. Repair differences	Change one place at a time

Three small experiments

- Only change the server name. Expectation: Only the corresponding row changes.
- Change print after "Player:" to println. Expectation: Alex will go to the next line.
- Add a blank line after the border. Expectation: The content remains unchanged, but the visual spacing changes.

When completed, explain each statement in your own words. Being able to change code and predict the results is more useful than copying it verbatim.

10 / 10

Take another small step forward

Select any work and write the expected output

- Basics: Write five lines of "My First Day of Survival" and use `println` at least three times.
- Advanced: First use `print` to spell out a line of synthetic formula, and then use two lines of explanation to explain the formula.
- Create: Draw a small house using text characters, at least seven lines high, checking the spaces in each line.

30 seconds self-test

- ① Will quotation marks appear in the output?
- ② Does the line break of `println` occur before or after the content?
- ③ Can the public class in `HelloWorld.java` be called `helloWorld`?

Answer: ① No, quotation marks define the text. ② After the content. ③ It cannot be changed directly like this; the public class name and file name must be consistent.

Next lesson: Let the program remember changes

Today's text is fixed. In the next lesson, we will label the health, item count, and player name so that the same name can save different values at different times.

Technical reference: Java's output API and language rules are available at docs.oracle.com. Examples are validated against Java 11; stories and illustrations are used to explain concepts.