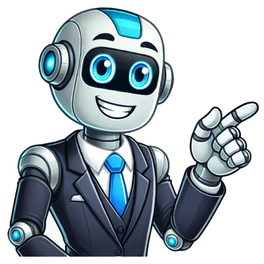
I'm not a robot



Practice java coding

If you haven't already, sign up to become a W3Schooler, and get points for every exercise you complete. As a logged-in W3Schools user you will have access to many features like having your own web page, track your learning progress, receive personal guided paths, and more. The Exercise The exercises are a mix of "multiple choice" and "fill in the blanks" questions. There are between 3 and 9 questions in each category. The answer can be found in the corresponding tutorial chapter. If you're stuck, or answer wrong, you can try again or hit the "Show Answer" button to see the correct answer. Get certified by completing the course Get certified w3schools CERTIFIED . 2025 Skip to content Accessibility Policy Oracle Java is the #1 programming language and development platform. It reduces costs, shortens development timeframes, drives innovation, and improves application services. With millions of developers running more than 60 billion Java Virtual Machines worldwide, Java continues to be the development platform of choice for enterprises and developers. Assess the health of your Java environment Java 24 is now available The next Java release improves the performance, stability, and security of Java application development. Oracle GraalVM free on OCI Build native executables that help applications start up fast, reduce memory usage, and save hosting costs. Java Management Service JMS Advance features help administrators gain additional insights into Java workloads. Analyze usage, vulnerabilities, and impact from Cryptographic Roadmap updates. Transforming Development for Next-Generation Software Supply Chains Get executive insights on leading trends and challenges impacting development organizations today, along with technology choices such as Java, that can help address them efficiently based on a recent VDC Research study. Get the VDC Research report Top IT security and compliance pain points in application development Security remains the most important priority for IT executives, according to a recent report from 451 Research. Read the brief to find out the top security concerns for developers and how you can achieve your IT security and compliance goals with Java. Get the 451 Research brief Aberdeen Knowledge Brief Learn why leaders in application development use Java to more efficiently build world-class applications the highest quality and the strongest security. Case study: Oracle Java EPP for Oracle Fusion Find out why Oracle Java Enterprise Performance Pack helps Oracle Fusion Applications improve application response times by 40% and decrease CPU utilization by 25%. Read the case study (PDF) Take advantage of the high-performance JDK with advanced optimizations that improve Java application performance and microservices deployment—on-premises and in the cloud. Together, the two technologies add value for cloud native deployments with native image and multilanguage support. Features Advanced optimizing just-in-time compiler Ahead-of-time compiler (compile Java native executables) Seamless interoperability for polyglot applications Built on enterprise-class Oracle Java SE 24/7 Oracle Premiere Support (My Oracle Support) Oracle Cloud Infrastructure (OCI) enhances the versatility, power, and stability of Java. As the steward and leading contributor to the Java platform, Oracle continues to drive the evolution of Java in response to the demands of enterprises and to provide unparalleled expertise to support developers. Explore Java on Oracle Cloud Infrastructure Enhanced Java performance. Boost peak performance with high-performance JDK. Build and deploy native executables that start almost instantly and use fewer memory resources. Simplified Java Management Service gives you insights into all your Java deployments, on OCI and on-premises. Expert Java support from the stewards of Java to optimize Java apps whenever it makes sense. Exclusive access to patches and updates, even beyond the end of public updates. Benefits of Java SE on OCI at no additional cost. Java Card enables secure elements, such as smart cards and other tamper-resistant security chips, to host applications based on Java technology. Store and update multiple applications on a single, resource-constrained device. Features Interoperable Secure execution environment Multiapplication, multitenant Extensible and updatable “One of our largest customers was facing performance and memory issues after upgrading their system. There was huge pressure from the customer at all levels to resolve the issue as soon as possible. Once the Java Sustaining Engineering team got involved, collaborating with Oracle Support, they were able to pinpoint what was causing the issue and what changes to make in order to resolve the issue. This solution was provided very quickly, and we received kudos from the end user’s executives as to the speed and efficacy of the solution. We greatly appreciate the efforts of the Java Sustaining Engineering Team and Oracle Customer Support.” Takashi Hashizume, Senior Manager, AI Platform Division, NEC Corporation MIKS Limited Computer Software Company "Oracle Java SE Subscription's multilingual support team is very experienced and readily available to provide instant and parallel support helping our developers to build their projects more timely and easily with no hassles." —Mohammad Iqbal Khan, Project Manager Rothbadi & Co. IT Services "Instead of wasting time and money, we have been able to reduce overall costs by managing our Java estate with Oracle Java SE Subscription. This gives us a huge cost saving opportunity that significantly reduces our IT OPEX bills." —Fortune Nwaiwu, Business Analyst "For our professional customers, where reliability is a top priority, the small fee of Oracle Jave SE subscription is much more valuable than the many free platforms available for the returned value. We use it for many deployments, mostly for high-value solutions, where every small detail matters." —Balázs Kiss, Software Developer Corte Suprema de Justicia "The technical support Oracle provides is highly efficient and of very good quality. Their staff is trained and has the necessary experience to solve or guide in the resolution of problems raised." —Moris Mendez, Ing. de Sistemas Informaticos March 18, 2025 Sharat Chander | Senior Director, Java Product Management and Developer Relations Learn about the Java 24 release and ongoing Java innovations that address modern application development. Read the complete post | Subscribe to the blog Manage Java SE installations, updates, and upgrades across your enterprise more cost effectively. Discover the advantages of a Java license and support from the Java experts who wrote the code. Talk to a Java team member about the advantages of an Oracle Java SE Subscription. Last update on March 13 2025 12:00:13 (UTC/GMT +8 hours) This resource offers a total of 5356 Java Programming problems for practice. It includes 1129 main exercises, each accompanied by solutions, detailed explanations, and 4 to 5 related problems. Java Exercises: Java is the backbone of networked, mobile, and enterprise applications, used by over 9 million developers worldwide. Practice exercises-from basic to advanced-with sample solutions to boost your coding skills. Challenge yourself, learn by doing, and enjoy coding! Note: If you are not habituated with Java programming you can learn from the following : Java Programming Language More to Come ! List of Exercises with Solutions : HTML CSS Exercises, Practice, Solution JavaScript Exercises, Practice, Solution jQuery Exercises, Practice, Solution jQuery-UI Exercises, Practice, Solution CoffeeScript Exercises, Practice, Solution Twitter Bootstrap Exercises, Practice, Solution C Programming Exercises, Practice, Solution C# Sharp Programming Exercises, Practice, Solution PHP Exercises, Practice, Solution Python Exercises, Practice, Solution R Programming Exercises, Practice, Solution Java Exercises, Practice, Solution SQL Exercises, Practice, Solution MySQL Exercises, Practice, Solution PostgreSQL Exercises, Practice, Solution SQLite Exercises, Practice, Solution MongoDB Exercises, Practice, Solution [Want to contribute to Java exercises? Send your code (attached with a .zip file) to us at w3resource[at]yahoo[dot]com. Please avoid copyrighted materials.] Do not submit any solution of the above exercises at here, if you want to contribute go to the appropriate exercise page. Warmup-1 Simple warmup problems to get started (solutions available) Warmup-2 Medium warmup string/array loops (solutions available) String-1 Basic string problems -- no loops Array-1 Basic array problems -- no loops. Logic-1 Basic boolean logic puzzles -- if else && || ! Logic-2 Medium boolean logic puzzles -- if else && || ! String-2 Medium String problems -- 1 loop String-3 Harder String problems -- 2 loops Array-2 Medium array problems -- 1 loop Array-3 Harder array problems -- 2 loops, more complex logic AP-1 AP CS medium problems Recursion-1 Basic recursion problems Recursion-2 Harder recursion problems Map-1 Basic Map get()/put(), no loops Map-2 Maps with bulk data and loops Functional-1 Functional mapping operations on lists with lambdas Functional-2 Functional filtering and mapping operations on lists with lambdas Misc Code Practice Seems like cookies are disabled on this browser, please enable them to open this website Codus has a library of hundreds of hand-written problems designed to reinforce programming skills. Codus saves your solutions as soon as you write them, so you never lose any work, ever. Codus checks every solution against a thorough set of test cases. Codus displays both stdout and stderr after every problem run in order to allow robust debugging. class Greeter { String getGreeting() { return "Hello, World!"; } } Java is a modern, fast-evolving language with releases every 6 months. Every expression has a type known at compile time. Java is primarily an object-oriented language, but has many functional features introduced in v1.8. Java is used for a variety of workloads like web, cloud, mobile and game applications. Java was designed to be cross-platform with the slogan "Write once, run anywhere". Java programs perform automatic memory management for their lifecycles. See all the concepts for Java The best part, it’s 100% free for everyone. Share — copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt — remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution — You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. Understanding Java fundamentals is the first step to becoming a proficient Java programmer. This collection of Java basic coding practice problems covers essential topics such as input/output operations, arithmetic and logical operators, type conversion, conditional statements, loops, and more. These exercises are categorized into Basic, Easy, and Medium levels, allowing you to gradually strengthen your Java programming skills. Practicing these problems will help build a strong foundation in Java syntax, logic building, and problem-solving, making it easier to tackle advanced programming challenges and technical interviews.Java Basics Practice QuestionsBasic:Easy:Mastering Java fundamentals is essential for building strong problem-solving skills and progressing to data structures, algorithms, and object-oriented programming. Practicing these Java coding problems will improve your logic, syntax understanding, and programming confidence, preparing you for technical interviews, This is a free set of tasks for your Java practice by CodeGym. If you’re a beginner, you can start learning the basics and get immediate feedback on your progress. If you’re a seasoned learner, it will help you estimate your current level of knowledge with additional Java challenges. Try to solve the first tasks – you’ll enjoy it! We’ve prepared a collection of Java exercises that will help you grasp the syntax of Java language and some core programming topics. In addition, you’ll find useful links to articles that cover the theory of Java. Enjoy your Java practice online and enhance your theory knowledge here! In this free simulator, you’ll find Java programming exercise with solutions verification. Just open the task, read the conditions, type your solution, and click “Verify”. You’ll get the result in a blink of an eye. There are different types of coding challenges in Java: writing your own code, correcting the existing one, and retyping, so your Java practice will be enjoyable and versatile. Tons of versatile Java coding tasks for learners with any background: from Java Syntax and Core Java topics to Multithreading and Java Collections The support from the CodeGym team and the global community of learners (“Help” section, programming forum, and chat) The modern tool for coding practice: with an automatic check of your solutions, hints on resolving the tasks, and advice on how to improve your coding style In Java programming, commands are essential instructions that tell the computer what to do. These commands are written in a specific way so the computer can understand and execute them. Every program in Java is a set of commands. At the beginning of your Java programming practice, it’s good to know a few basic principles: In Java, each command ends with a semicolon; A command can't exist on its own: it’s a part of a method, and method is part of a class; Method (procedure, function) is a sequence of commands. Methods define the behavior of an object. Here is an example of the command: System.out.println("Hello, World!"); The command System.out.println("Hello, World!"); tells the computer to display the text inside the quotation marks. If you want to display a number and not text, then you do not need to put quotation marks. You can simply write the number. Or an arithmetic operation. For example: System.out.println(1); Command to display the number 1. System.out.println(5 + 5); A command in which two numbers are summed and their sum (10) is displayed. As we discussed in the basic rules, a command cannot exist on its own in Java. It must be within a method, and a method must be within a class. Here is the simplest program that prints the string "Hello, World!". public class HelloWorld { public static void main(String[] args) { //here we print the text out System.out.println("Hello, World!"); } } We have a class called HelloWorld, a method called main(), and the command System.out.println("Hello, World!"). You may not understand everything in the code yet, but that's okay! You'll learn more about it later. The good news is that you can already write your first program with the knowledge you've gained. Attention! You can add comments in your code. Comments in Java are lines of code that are ignored by the compiler, but you can mark with them your code to make it clear for you and other programmers. Single-line comments start with two forward slashes (//) and end at the end of the line. In example above we have a comment //here we print the text out You can read the theory on this topic here, here, and here. But try practicing first! Explore the Java coding exercises for practicing with commands below. First, read the conditions, scroll down to the Solution box, and type your solution. Then, click Verify (above the Conditions box) to check the correctness of your program. Exercise 1 Exercise 2 Exercise 3 Start task Types and keyboard input in Java The two main types in Java are String and int. We store strings/text in String, and integers (whole numbers) in int. We have already used strings and integers in previous examples without explicit declaration, by specifying them directly in the System.out.println() operator. System.out.println("I am a String"); System.out.println(5); In the first case “I am a string” is a String in the second case 5 is an integer of type int. However, most often, in order to manipulate data, variables must be declared before being used in the program. To do this, you need to specify the type of the variable and its name. You can also set a variable to a specific value, or you can do this later. Example: int a; int b = 5; String s = "Hello, World!"; Here we declared a variable called a but didn't give it any value, declared a variable b and gave it the value 5, declared a string called s and gave it the value Hello, World! Attention! In Java, the = sign is not an equals sign, but an assignment operator. That is, the variable (you can imagine it as an empty box) is assigned the value that is on the right (you can imagine that this value was put in the empty box). int a; a = 5; We created an integer variable named a with the first command and assigned it the value 5 with the second command. Before moving on to practice, let's look at an example program where we will declare variables and assign values to them: public class HelloWorld { public static void main(String[] args) { int a; int b = 5; String s = “Hello, World!”; a = 2; System.out.println(a); System.out.println(a + b); System.out.println(s); } } In the program, we first declared an int variable named a but did not immediately assign it a value. Then we declared an int variable named b and "put" the value 5 in it. Then we declared a string named s and assigned it the value "Hello, World!" After that, we assigned the value 2 to the variable a that we declared earlier, and then we printed the variable a, the sum of the variables a and b, and the variable s to the screen This program will display the following: 2 7 Hello, World! We already know how to print to the console, but how do we read from it? For this, we use the Scanner class. To use Scanner, we first need to create an instance of the class. We can do this with the following code: Scanner scanner = new Scanner(System.in); Once we have created an instance of Scanner, we can use the next() method to read input from the console or nextInt() if we should read an integer. The following code reads a number from the console and prints it to the console: import java.util.Scanner; public class Main { public static void main(String[] args) { System.out.println("Enter a number "); Scanner scanner = new Scanner(System.in); int number = scanner.nextInt(); System.out.println(number); } } Here we first import a library scanner, then ask a user to enter a number. Later we created a scanner to read the user's input and print the input out. This code will print the following output in case of user’s input is 5: Enter a number 5 5 More information about the topic you could read here, here, and here. See the exercises on Types and keyboard input to practice Java coding: Exercise 1 Exercise 2 Exercise 3 Start task Conditions and If statements in Java Conditions and If statements in Java allow your program to make decisions. For example, you can use them to check if a user has entered a valid password, or to determine whether a number is even or odd. For this purpose, there’s an 'if/else statement' in Java. The syntax for an if statement is as follows: if (condition1) { // code to be executed if the condition is true }... if (conditionN) { // code to be executed if the condition is true } else { // code to be executed if the conditions is false } Here could be one or more conditions in if and zero or one condition in else. Here's a simple example: int age = 18; if (age >= 18) { System.out.println("You are an adult."); } else { System.out.println("You are a minor."); } In this example, we check if the variable "age" is greater than or equal to 18. If it is, we print "You are an adult." If not, we print "You are a minor." More information about the topic you could read here, here, and here. Here are some Java practice exercises to understand Conditions and If statements: Exercise 1 Exercise 2 Exercise 3 Start task In Java, a "boolean" is a data type that can have one of two values: true or false. Here's a simple example: boolean isJavaFun = true; boolean isCodingEasy = false; System.out.println("Is Java fun? " + isJavaFun); System.out.println("Is coding easy? " + isCodingEasy); The output of this program is here: Is Java fun? true Is coding easy? false In addition to representing true or false values, booleans in Java can be combined using logical operators. Here, we introduce the logical AND (&&) and logical OR (||) operators. boolean isJavaFun = true; boolean isCodingEasy = false; // Using logical operators boolean isBothFunAndEasy = isJavaFun && !isCodingEasy; boolean isEitherFunOrEasy = isJavaFun || isCodingEasy; System.out.println("Is it both fun and easy? " + isBothFunAndEasy); System.out.println("Is it either fun or easy? " + isEitherFunOrEasy); && (AND) returns true if both operands are true. In our example, isBothFunAndEasy is true because Java is fun (isJavaFun is true) and coding is not easy (isCodingEasy is false). || (OR) returns true if at least one operand is true. In our example, isEitherFunOrEasy is true because Java is fun (isJavaFun is true), even though coding is not easy (isCodingEasy is false). The NOT operator (!) is unary, meaning it operates on a single boolean value. It negates the value, so !isCodingEasy is true because it reverses the false value of isCodingEasy. So the output of this program is: Is it both fun and easy? true Is it either fun or easy? true More information about the topic you could read here, and here. Here are some Java exercises to practice booleans: Start task With loops, you can execute any command or a block of commands multiple times. The construction of the while loop is: Loops are essential in programming to execute a block of code repeatedly. Java provides two commonly used loops: while and for. 1. while Loop: The while loop continues executing a block of code as long as a specified condition is true. Firstly, the condition is checked. While it’s true, the body of the loop (commands) is executed. If the condition is always true, the loop will repeat infinitely, and if the condition is false, the commands in a loop will never be executed. Example: int count = 1; while (count In this example, the code inside the while loop will run repeatedly as long as count is less than or equal to 5. 2. for Loop: The for loop is used for iterating a specific number of times. Example: for (int i = 1; i In this for loop, we initialize i to 1, specify the condition i More information about the topic you could read here, here, and here. Here are some Java coding challenges to practice the loops: Exercise 1 Exercise 2 Exercise 3 Start task An array in Java is a data structure that allows you to store multiple values of the same type under a single variable name. It acts as a container for elements that can be accessed using an index. What you should know about arrays in Java: Indexing: Elements in an array are indexed, starting from 0. You can access elements by specifying their index in square brackets after the array name, like myArray[0] to access the first element. Initialization: To use an array, you must declare and initialize it. You specify the array's type and its length. For example, to create an integer array that can hold five values: int[] myArray = new int[5]; Populating: After initialization, you can populate the array by assigning values to its elements. All elements should be of the same data type. For instance, myArray[0] = 10; myArray[1] = 20;. Default Values: Arrays are initialized with default values. For objects, this is null, and for primitive types (int, double, boolean, etc.), it's typically 0, 0.0, or false. Example: // Declare and initialize an integer array of length 3 int[] myArray = new int[3]; // Assign values to array elements myArray[0] = 10; myArray[1] = 20; myArray[2] = 30; // Access and print an element System.out.println("Element at index 1: " + myArray[1]); In this example, we create an integer array, assign values to its elements, and access an element using indexing. More information about the topic you could read here, here, and here. Exercise 1 Exercise 2 Exercise 3 Start task Working with methods in Java In Java, methods are like mini-programs within your main program. They are used to perform specific tasks, making your code more organized and manageable. Methods take a set of instructions and encapsulate them under a single name for easy reuse. Here's how you declare a method: public static void methodName() { // Method body with commands } public is an access modifier that defines who can use the method. In this case, public means the method can be accessed from anywhere in your program.Read more about modifiers here. static means the method belongs to the class itself, rather than an instance of the class. It's used for the main method, allowing it to run without creating an object. void indicates that the method doesn't return any value. If it did, you would replace void with the data type of the returned value. Examples: public class MethodExample { public static void main(String[] args) { // Call the customMethod customMethod(); } public static void customMethod() { // This is our custom method System.out.println("Hello from customMethod!"); } } In this example, we have a main method (the entry point of the program) and a customMethod that we've defined. The main method calls customMethod, which prints a message. This illustrates how methods help organize and reuse code in Java, making it more efficient and readable. public class Main { public static void main(String[] args) { int result = add(5, 3); // Calling the add method System.out.println("The sum is: " + result); } public static int add(int num1, int num2) { int sum = num1 + num2; return sum; } } In this example, we have a main method that calls the add method with two numbers (5 and 3). The add method calculates the sum and returns it. The result is then printed in the main method. More information about the topic you could read here, here, and here. Exercise 1 Exercise 2 Exercise 3 Start task All composite types in Java consist of simpler ones, up until we end up with primitive types. An example of a primitive type is int, while String is a composite type that stores its data as a table of characters (primitive type char). Here are some examples of primitive types in Java: int: Used for storing whole numbers (integers). Example: int age = 25; double: Used for storing numbers with a decimal point. Example: double price = 19.99; char: Used for storing single characters. Example: char grade = 'A'; boolean: Used for storing true or false values. Example: boolean isJavaFun = true; String: Used for storing text (a sequence of characters). Example: String greeting = "Hello, World!"; Simple types are grouped into composite types, that are called classes. Example: public class Person { String name; int age; } We declared a composite type Person and stored the data in a String (name) and int variable for an age of a person. Since composite types include many primitive types, they take up more memory than variables of the primitive types. More information about the topic you could read here, here, and here. See the exercises for a coding practice in Java data types: Exercise 1 Exercise 2 Exercise 3 Start task Working with strings in Java String is the most popular class in Java programs. Its objects are stored in a memory in a special way. The structure of this class is rather simple: there’s a character array (char array) inside, that stores all the characters of the string. String class also has many helper classes to simplify working with strings in Java, and a lot of methods. Here’s what you can do while working with strings: compare them, search for substrings, and create new substrings. Example of comparing strings using the equals() method. String str1 = "Hello"; String str2 = "Hello"; if (str1.equals(str2)) { System.out.println("Strings are equal."); } else { System.out.println("Strings are not equal."); } Also you can check if a string contains a substring using the contains() method. String text = "This is a sample text."; String search = "sample"; if (text.contains(search)) { System.out.println("Substring found."); } else { System.out.println("Substring not found."); } You can create a new substring from an existing string using the substring() method. String text = "Hello, World!"; String subText = text.substring(7); System.out.println(subText); // Output: "World!" More information about the topic you could read here, here, here, here, and here. Here are some Java programming exercises to practice the strings: Start task In Java, objects are instances of classes that you can create to represent and work with real-world entities or concepts. Here's how you can create objects: First, you need to define a class that describes the properties and behaviors of your object. You can then create an object of that class using the new keyword like this: ClassName objectName = new ClassName(); It invokes the constructor of a class.If the constructor takes arguments, you can pass them within the parentheses. For example, to create an object of class Person with the name "Jane" and age 25, you would write: Person john = new Person("Jane", 25); Example: Suppose you want to create a simple Person class with a name property and a sayHello method. Here's how you do it: // Step 1: Define the Person class class Person { String name; // Property // Method to say hello void sayHello() { System.out.println("Hello, my name is " + name); } } public class Main { public static void main(String[] args) { // Step 2: Create an object of the Person class Person person1 = new Person(); // Set the name property person1.name = "Johnny"; // Call the sayHello method person1.sayHello(); // Output: "Hello, my name is Johnny" // Create another object Person person2 = new Person(); person2.name = "Ivy"; person2.sayHello(); // Output: "Hello, my name is Ivy" } } In this example, we defined a Person class with a name property and a sayHello method. We then created two Person objects (person1 and person2) and used them to represent individuals with different names. More information about the topic you could read here, here, and here. Here are some coding challenges in Java object creation: Exercise 1 Exercise 2 Exercise 3 Start task Static Classes and Methods in Java Static classes and methods in Java are used to create members that belong to the class itself, rather than to instances of the class. They can be accessed without creating an object of the class. Static methods and classes are useful when you want to define utility methods or encapsulate related classes within a larger class without requiring an instance of the outer class. They are often used in various Java libraries and frameworks for organizing and providing utility functions. You declare them with the static modifier. Static Methods A static method is a method that belongs to the class rather than any specific instance. You can call a static method using the class name, without creating an object of that class. public class Calculator { public static int add(int num1, int num2) { return num1 + num2; } } public class Main { public static void main(String[] args) { int result = Calculator.add(5, 3); System.out.println("Result: " + result); // Output: Result: 8 } } In this example, the add method is static. You can directly call it using Calculator.add(5, 3) Static Classes In Java, you can also have static nested classes, which are classes defined within another class and marked as static. These static nested classes can be accessed using the outer class's name. public class School { static class Student { String name; Student(String name) { this.name = name; } } } public class Main { public static void main(String[] args) { School.Student student1 = new School.Student("Johnny"); School.Student student2 = new School.Student("Ivy"); System.out.println("Student 1: " + student1.name); // Output: Student 1: Johnny System.out.println("Student 2: " + student2.name); // Output: Student 2: Ivy } } In this example, Student is a static nested class within the School class. You can access it using School.Student. More information about the topic you could read here, here, here, and here. See below the exercises on Static classes and methods in our Java coding practice for beginners: Exercise 1 Exercise 2 Exercise 3 Start task