



Elysium Academy Spark Notes

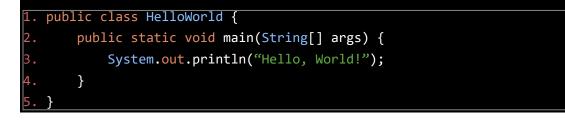
VERSION 2.4



01. Java Basics

Java is a versatile and widely-used programming language that supports Object-Oriented Programming (OOP) principles. It is known for its portability across platforms (write once, run anywhere), robustness, and security.

Hello World Program



- Key Points:
 - Class Declaration: Java programs are organized into classes, which contain the code. The keyword class defines a class.
 - main() Method: The entry point of any Java application. It has the signature public static void main(String[] args).
 - System.out.println(): A method to print output to the console.

02. Data Types

Java provides a wide range of data types, which are divided into primitive and non-primitive data types.

Primitive Data Types

Data Type	Size	Default Value	Example Usage
byte	1 byte	0	byte b = 100;
short	2 bytes	0	short s = 5000;
int	4 bytes	0	int num = 12345;
long	8 bytes	OL	long I = 123456789L;
float	4 bytes	0.0f	float f = 5.75f;
double	8 bytes	0.0d	double d = 19.99;
char	2 bytes	\u0000	char letter = 'A';
boolean	1 bit	false	boolean flag = true;



• Non-Primitive Data Types

Туре	Description	Example Usage
String	Sequence of characters (class)	String name = "Java";
Array	A fixed-size collection of values	int[] nums = {1, 2};
Class	User-defined object	Defined by developers

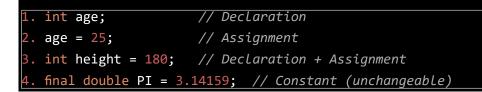
Example :

- 1. int age = 30;
- 2. String name = "John";
- // Primitive data type
 // Non-primitive data type (String class)
- 3. final int DAYS_IN_WEEK = 7; // Constant variable

03. Variables

Variables are containers for storing data. Java is a statically-typed language, meaning each variable must be declared with a data type.

Declaring Variables



• Variable Types:

- Local Variables: Declared inside methods, constructors, or blocks and used only within that scope.
- Instance Variables: Declared inside a class but outside methods, representing the state of an object.
- Static/Class Variables: Declared with the static keyword; shared among all objects of a class.

04. Operators



Java has various operators to perform operations on variables and values.

• Arithmetic Operators:

Operator	Description	Example
+	Addition	a + b
	Subtraction	a - b
* Multiplication		a*b
1	Division	a/b
%	Modulus (remain- der)	a % b

Relational/Comparison Operators:

Operator	Description	Example
==	Equal to	a == b
!=	Not equal to a	
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal	a >= b
<=	Less than or equal	a <= b

• Logical Operators:

Operator	Description	Example
&&	Logical AND	a && b
、 、		、 、
!	Logical NOT	!a

Assignment Operators:

Operator	Description	Example
=	Assign value	a = b;
+= Add and assign		a += b;
-= Subtract and as- sign a -= b;		a -= b;
*=	Multiply and assign	a *= b;
/=	Divide and assign	a /= b;

Increment/Decrement Operators:



Operator	Description	Example
<pre>(++);;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;</pre>	Increment by 1	a++ or ++a
· · · · · · · · · · · · · · · · · · ·	Decrement by 1	a ora

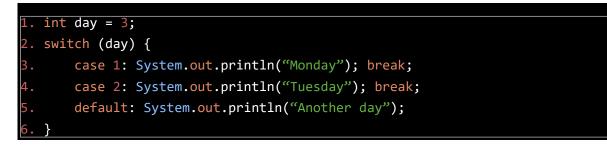
05. Control Flow Statements

Control flow statements determine the flow of execution in a Java program. These include conditional statements, loops, and branching statements.

• If-Else Statement:



• Switch Case:



• Loops

• For Loop:

```
1. for (int i = 0; i < 5; i++) {
2. System.out.println(i);</pre>
```

While Loop:

```
1. int i = 0;
2. while (i < 5) {
3. System.out.println(i);
4. i++;
5. }
```





• Do-While Loop:

1. int i = 0; 2. do { 3. System.out.println(i); 4. i++; 5. } while (i < 5);</pre>

Branching Statements:

Statement	Description	Example
break	Exits a loop or switch statement	break;
continue Skips the current iteration		continue;
return	Exits the current method and returns a value	return value;

06. Arrays

Control flow statements determine the flow of execution in a Java program. These include conditional statements, loops, and branching statements.

• Array Declaration:

1. int[] numbers = {1, 2, 3, 4, 5};	// Array Initialization
<pre>2. int[] scores = new int[5];</pre>	// Declaring an empty array of size 5

• Accessing Array Elements:

1. int firstNumber = numbers[0]; // Access first element
2. numbers[2] = 10; // Modify third element

• Iterating through Arrays:

```
1. for (int num : numbers) {
2. System.out.println(num);
3. }
```

Multidimensional Arrays:

L. int[][] matrix = { {1, 2, 3}, {4, 5, 6} }; // 2D Array
2. int value = matrix[0][1]; // Access element at row 0, column 1

C Copyrights by Elysium Academy Private Limited



07. Methods (Functions)

Methods allow reusability and modularity by grouping code into reusable blocks. Java supports both predefined methods (e.g., System.out.println()) and user-defined methods.

• Defining Methods:

```
1. public static int add(int a, int b) {
2. return a + b;
```

• Calling Methods:

```
1. int sum = add(5, 3); // Method call
2. System.out.println(sum); // Output: 8
```

Method Overloading:

Method overloading allows methods to have the same name but different parameter lists (type, number, or order of parameters).

```
1. public int add(int a, int b) { return a + b; }
2. public double add(double a, double b) { return a + b; }
```

08. Object-Oriented Programming (OOP)

Java is an object-oriented programming language, which revolves around the following key concepts:

8.1. Classes and Objects

A class is a blueprint for creating objects. An object is an instance of a class.

Class Definition:

1. (class Dog {	•
± . (
2.	String name;	•
3.	int age;	
4.	<pre>void bark() {</pre>	•
5.	<pre>System.out.println("Woof!");</pre>	•
6.	}	•••••
7.	}	



Creating an Object:

1.	<pre>Dog myDog = new Dog();</pre>	// Creating an object
2.	<pre>myDog.name = "Buddy";</pre>	// Accessing fields
3.	myDog.bark();	// Calling methods

8.2. Constructors

A constructor is a special method used to initialize objects. It has the same name as the class and no return type.

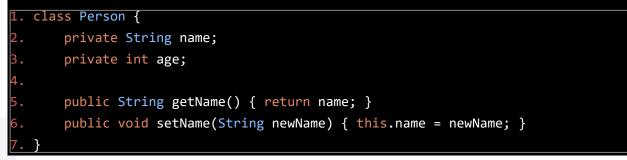
```
1. class Dog {
2. String name;
3. int age;
4.
5. // Constructor
6. Dog(String name, int age) {
7. this.name = name;
8. this.age = age;
9. }
10. }
```

• Creating an Object with Constructor:

1. Dog myDog = new Dog("Buddy", 3);

8.3. Encapsulation

Encapsulation is the concept of bundling data (fields) and methods that operate on that data into a single unit (class). The fields are made private, and access is provided through public getters and setters.



8.4. Inheritance

Inheritance allows a new class to inherit properties and behaviors (methods) from an existing class. The class being inherited from is called the superclass (parent), and the class inheriting is the subclass (child).

 $\ensuremath{\mathbb{C}}$ Copyrights by Elysium Academy Private Limited



1. class Ar	nimal {
2. void	eat() {
3.	<pre>System.out.println("This animal eats food.");</pre>
4. }	
5. }	
6.	
7. class Do	og extends Animal {
8. void	bark() {
9.	<pre>System.out.println("Woof!");</pre>
10. }	
11. }	

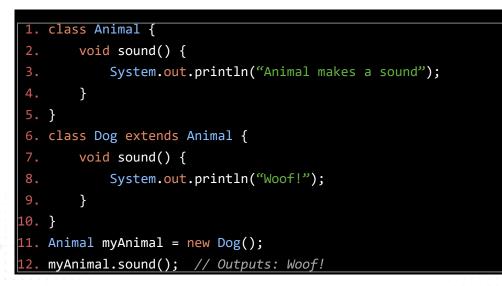
• Using Inheritance:

1.	Dog myDog = new	Dog();
2.	<pre>myDog.eat();</pre>	// Inherited from Animal class
3.	<pre>myDog.bark();</pre>	// Defined in Dog class

8.5. Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It can occur through method overriding and method overloading.

• Method Overriding:



8.6. Abstraction

Abstraction is the concept of hiding the internal details of an implementation and only exposing the essential features. In Java, abstraction is achieved using abstract classes or interfaces.





Abstract Class:



09. Exception Handling

Java uses exceptions to handle runtime errors. Exceptions are objects that represent an error. Java uses the try-catch-finally mechanism for handling exceptions.

• Try-Catch Block:

```
1. try {
2. int result = 10 / 0; // This will throw ArithmeticException
3. } catch (ArithmeticException e) {
4. System.out.println("Cannot divide by zero!");
5. } finally {
6. System.out.println("This block always executes.");
7. }
```

Throwing Exceptions:

```
1. if (age < 18) {
2. throw new IllegalArgumentException("Age must be 18 or older.");
3. }</pre>
```

Types of Exceptions:

• **Checked Exceptions:** Exceptions checked at compile-time (e.g., IOException, SQLException).

• Unchecked Exceptions: Exceptions not checked at compile-time but at runtime (e.g., NullPointerException, ArrayIndexOutOfBounds-Exception).



10. File Handling

File handling in Java is done using classes from the java.io package, such as File, Scanner, FileWriter, BufferedReader, etc.

• Reading from a File:

```
1. import java.io.File;
```

- 2. import java.io.FileNotFoundException;
- 3. import java.util.Scanner;
- 4. File file = new File("filename.txt");
- 5. Scanner reader = new Scanner(file);
- 6. while (reader.hasNextLine()) {
- 7. String data = reader.nextLine();
- 8. System.out.println(data);

```
9. }
```

10. reader.close();

• Writing to a File:

```
1. import java.io.FileWriter;
2. import java.io.IOException;
3. FileWriter writer = new FileWriter("filename.txt");
```

```
    writer.write("Hello, File!");
```

- 5. writer.close();
- Handling IOException:

1. try { 2. FileWriter writer = new FileWriter("filename.txt"); 3. writer.write("Hello, File!"); 4. writer.close(); 5. } catch (IOException e) { 6. System.out.println("An error occurred."); 7. e.printStackTrace(); 8 }

11. Packages and Imports

A package in Java is a namespace that organizes a set of related classes and interfaces.





• Using Built-In Packages:

```
1. import java.util.Scanner;
```

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

```
3. int age = input.nextInt();
```

• Creating a Custom Package:

```
1. package myPackage;
```

```
2. public class MyClass {
```

```
. public void displayMessage() {
```

- System.out.println("Hello from MyClass");
- Using the Custom Package:

```
1. import myPackage.MyClass;
2. public class Main {
3. public static void main(String[] args) {
4. MyClass obj = new MyClass();
5. obj.displayMessage();
6. }
7. }
```

12. Useful Java Libraries

Java comes with a vast set of libraries, also called APIs (Application Programming Interfaces), that provide a wide variety of functions.

12.1. String Manipulation:

- length(): Returns the length of the string.
- charAt(index): Returns the character at the specified index.
- substring(start, end): Returns the substring.
- indexOf(char): Returns the index of the first occurrence of the character.

```
1. String str = "Hello, Java!";
2. System.out.println(str.length()); // Output: 12
3. System.out.println(str.charAt(0)); // Output: H
4. System.out.println(str.substring(0, 5)); // Output: Hello
5. System.out.println(str.indexOf('J')); // Output: 7
```





12.2. Collections Framework:

Java's Collections Framework provides data structures like Lists, Sets, Maps, etc.

```
1. import java.util.ArrayList;
```

```
2. import java.util.HashMap;
```

```
3. ArrayList<String> names = new ArrayList<>();
```

```
4. names.add("Alice");
```

```
5. names.add("Bob");
```

```
6. HashMap<String, Integer> ages = new HashMap<>();
```

```
7. ages.put("Alice", 30);
```

```
8. ages.put("Bob", 25);
```

12.3. Date and Time API:

Java 8 introduced the java.time package, which provides better date and time manipulation capabilities.

```
1. import java.time.LocalDate;
```

```
2. import java.time.LocalTime;
```

```
3. LocalDate date = LocalDate.now();
```

```
4. LocalTime time = LocalTime.now();
```

```
5. System.out.println("Current Date: " + date);
```

```
6. System.out.println("Current Time: " + time);
```

13. Java Annotations

Annotations provide metadata about the code and do not change the program's actual logic.

Common Annotations:

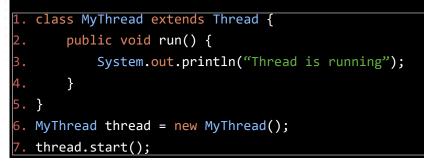
- @Override: Indicates that a method is overriding a superclass method.
- **@Deprecated:** Marks a method as deprecated (i.e., no longer recommended for use).
- @SuppressWarnings: Suppresses compiler warnings.

14. Java Threads and Concurrency



Java provides built-in support for multithreading and concurrent programming through the Thread class and the Runnable interface.

- Creating a Thread:
 - By Extending the Thread Class:



• By Implementing Runnable Interface:



• Concurrency Utilities:

Java provides higher-level concurrency APIs like ExecutorService, Callable, Future, etc., for handling multiple threads efficiently.



15. Java Memory Management



Java provides automatic memory management with the help of Garbage Collection. The garbage collector reclaims memory used by objects that are no longer in use.

Memory Areas in Java:

- Heap: The runtime data area where objects are allocated.
- Stack: Stores method calls and local variables.
- **Method Area:** Contains class structure like metadata, methods, constants, and static variables.
- **PC Registers:** Program counter register, storing addresses of current instructions.

16. Java Design Patterns

Design patterns are reusable solutions to commonly occurring problems in software design. Some popular patterns in Java include:

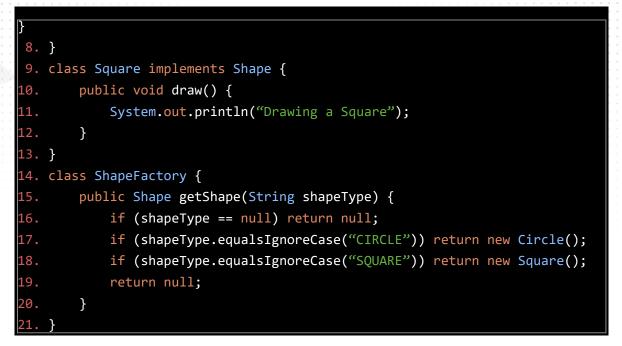
- Creational Patterns:
 - Singleton Pattern: Ensures that a class has only one instance.

```
1. public class Singleton {
2. private static Singleton instance;
3. private Singleton() { }
4. public static Singleton getInstance() {
5. if (instance == null) {
6. instance = new Singleton();
7. }
8. return instance;
9. }
10. }
```

• Factory Pattern: Provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.

```
1. interface Shape {
2. void draw();
3. }
4. class Circle implements Shape {
5. public void draw() {
```





Structural Patterns:

- Adapter Pattern: Allows incompatible interfaces to work together.
- **Decorator Pattern:** Allows behavior to be added to an individual object, dynamically.

• Behavioral Patterns:

- **Observer Pattern:** Defines a one-to-many dependency between objects.
- Strategy Pattern: Allows algorithms to be selected at runtime.

17. Java Advanced Topics

17.1. Java Streams (Java 8):

Streams are used to process collections of objects. They support operations like filtering, mapping, and reducing in a declarative way.

```
1. import java.util.Arrays;
2. import java.util.List;
3. List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
4. names.stream().filter(s -> s.startsWith("A")).forEach(System.out::print
ln); // Output: ALice
```

17.2. Lambda Expressions (Java 8):



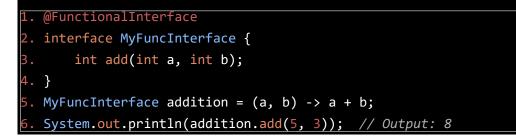


Lambdas are anonymous functions that provide a concise way to represent instances of functional interfaces.



17.3. Functional Interfaces (Java 8):

A functional interface has only one abstract method but can have multiple default or static methods. Some commonly used functional interfaces are Runnable, Callable, Supplier, Consumer, etc.



18. Java Best Practices

- Follow Naming Conventions:
 - Use camelCase for variables and methods (e.g., myVariable).
 - Use PascalCase for class names (e.g., MyClass).
 - Use ALL_CAPS for constants (e.g., MAX_VALUE).
- DRY Principle (Don't Repeat Yourself):
 - Write reusable code by avoiding redundancy.
- Code Documentation:
 - Use comments and Javadoc for better readability and maintainability.
- Error Handling:
 - Catch specific exceptions and avoid using broad exception classes (e.g., avoid catch(Exception e) unless necessary).

• Immutable Classes:

• Make classes immutable to improve thread safety.



• Avoid Memory Leaks:

• Be cautious about object references, particularly in long-lived objects such as collections.

18. Conclusion

Java is a powerful, high-level programming language that has withstood the test of time due to its robustness, security, portability, and performance. This comprehensive has covered the fundamental and advanced aspects of Java, offering a valuable reference to both beginners and seasoned developers alike. From core language features like data types, operators, and control flow statements, to advanced topics like OOP principles, file handling, concurrency, and design patterns, Java continues to evolve, supporting modern development paradigms like functional programming and streams.

As you continue learning Java, explore the vast ecosystem of libraries, frameworks, and tools that extend Java's capabilities. Whether you're building enterprise-level applications, Android apps, or working with big data technologies like Hadoop and Spark, Java remains a top choice for developers worldwide. Happy coding!

Click Here To Find Out More







elysiumacademy.org

info@elysiumacademy.org

Scan Here for More Spark Notes

