

Comprehensive Notes on: Fundamentals of Java Programming

Ashish Kumar Bhoi
Odisha Adarsha Vidyalaya Bibina

October 28, 2025

Contents

1	Comprehensive Notes	2
1.1	Introduction to Java	2
1.1.1	What is a Program?	2
1.1.2	Programming Languages: From Human to Machine	2
1.1.3	The Java Approach: The Best of Both Worlds	2
1.1.4	JDK, JRE, and JVM	3
1.2	Your First Java Program: "Hello World"	4
1.2.1	Anatomy of the Program	4
1.3	Data Types and Variables	5
1.3.1	Variables	5
1.3.2	Primitive Data Types	5
1.3.3	Reference Data Types (Non-Primitive)	5
1.4	Operators in Java	5
1.5	Control Flow Statements	6
1.5.1	Selection Statements	6
1.5.2	Repetition (Looping) Statements	6
1.6	Arrays	7
1.7	Object-Oriented Programming (OOP)	7
1.7.1	Class and Object	7
1.7.2	Creating a Class and Objects	7
1.7.3	Constructors	8
1.7.4	Access Modifiers and Encapsulation	8
1.8	Exception Handling	9
1.9	Database Connectivity (JDBC)	9
1.10	Assertions, Threads, and Wrapper Classes	10
1.10.1	Assertions	10
1.10.2	Threads	11
1.10.3	Wrapper Classes	11
2	Practice Questions	13
2.1	Multiple Choice Questions (MCQs)	13
2.2	One-Word Answer Questions	15
2.3	Differentiate Between Questions	16
2.4	Short Answer Questions	17
2.5	Long Answer Questions	18
2.6	Practical Application Questions	20

1 Comprehensive Notes

1.1 Introduction to Java

1.1.1 What is a Program?

A computer program is a set of instructions that we give to a computer to perform a specific task. Think of it like a recipe for a dish. The recipe has a list of ingredients (data) and a sequence of steps (instructions) to follow. Similarly, a program tells the computer exactly what to do and how to do it. The process of creating these programs is called **programming**.

1.1.2 Programming Languages: From Human to Machine

Programmers write programs using **high-level programming languages** like Java, Python, C++, etc. These languages are "high-level" because their syntax is closer to human language, making them easier to read, write, and understand.

However, a computer's processor can only understand one language: **machine language** (also called machine code). This is a "low-level" language consisting of binary code (0s and 1s).

To bridge this gap, we need a translator. There are two main types of translators:

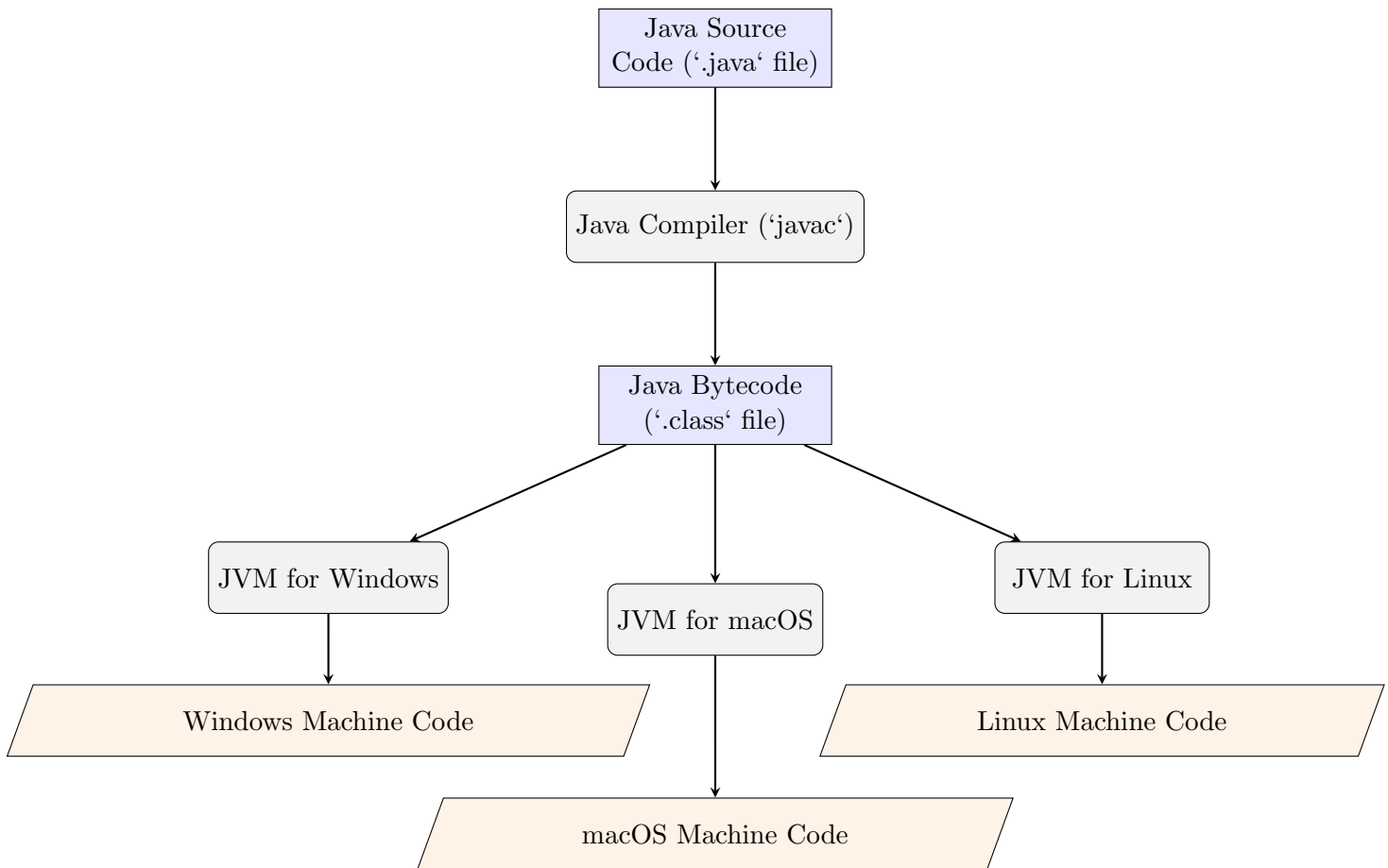
- **Compiler:** A compiler translates the *entire* high-level program into machine code at once. It creates a separate executable file (e.g., an '.exe' file on Windows). You then run this executable file. If there are any errors in your code, the compiler will report them all after trying to compile the whole program.
- **Interpreter:** An interpreter translates and executes the high-level program *line by line*. It does not create a separate executable file. If it finds an error, it stops at that line and reports it.

1.1.3 The Java Approach: The Best of Both Worlds

Java uses a unique, hybrid approach that combines both compilation and interpretation. This is the key to its famous slogan: **"Write Once, Run Anywhere" (WORA)**.

Here's how it works:

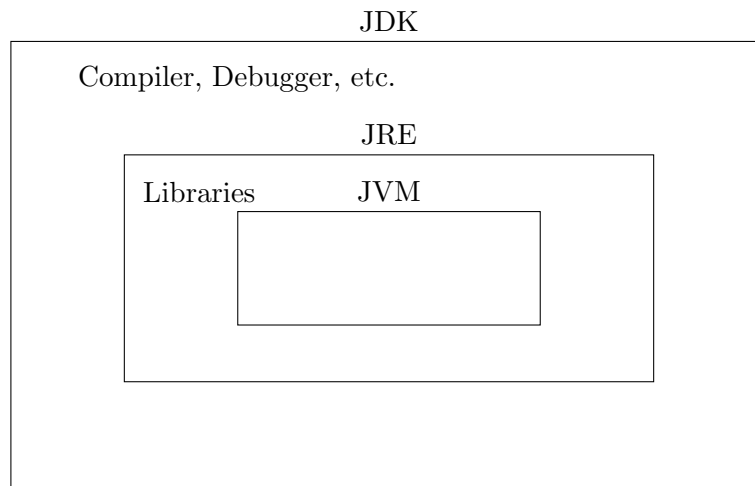
1. **Writing the Code:** You write your Java program in a text file with a '.java' extension. This is called the **source code**.
2. **Compilation:** You use the Java compiler ('javac') to compile your '.java' file. Instead of creating platform-specific machine code, the Java compiler creates a highly optimized, intermediate code called **Java Bytecode**. This bytecode is saved in a '.class' file.
3. **Execution:** This '.class' file (containing bytecode) can be run on any computer, regardless of its operating system (Windows, macOS, Linux), as long as it has a **Java Virtual Machine (JVM)** installed.
4. **The JVM:** The JVM is the "magic" component. It is a virtual machine that acts as an interpreter for the bytecode. When you run the program, the JVM reads the bytecode, translates it into the native machine code for the specific computer it's running on, and then executes it.



1.1.4 JDK, JRE, and JVM

These three acronyms are fundamental to the Java ecosystem.

- **JVM (Java Virtual Machine):** The core component that runs Java bytecode. It is platform-dependent, meaning there's a different JVM for Windows, macOS, etc.
- **JRE (Java Runtime Environment):** The software package that provides the JVM and other libraries needed to *run* Java applications. If you only want to run Java programs (not develop them), you only need the JRE.
- **JDK (Java Development Kit):** The full-featured software development kit for Java. It includes everything in the JRE, plus development tools like the compiler (.javac), debugger, and other utilities needed to *create* Java applications.



1.2 Your First Java Program: "Hello World"

Let's write a simple program to print "Hello World" to the screen. This is a tradition for learning any new programming language.

```
1 // This is a simple Java program.
2 // It prints "Hello World" to the console.
3
4 public class HelloWorld {
5     public static void main(String[ ] args) {
6         // The main method is the entry point of the program.
7         System.out.println("Hello World"); // Prints the text.
8     }
9 }
```

Listing 1: HelloWorld.java

1.2.1 Anatomy of the Program

Let's break down this program piece by piece.

- `'// ...'`: These are **comments**. The compiler ignores them. They are for humans to read and understand the code. Java has two types of comments:
 - Single-line comments start with `'//'`.
 - Multi-line comments start with `'/*'` and end with `'*/'`.
- `'public class HelloWorld'`: This line declares a **class** named `'HelloWorld'`.
 - `'class'`: A keyword used to define a class. A class is a blueprint for creating objects. All Java code must reside inside a class.
 - `'HelloWorld'`: The name of our class. By convention, class names start with a capital letter. The file must be saved as `'HelloWorld.java'`.
 - `'public'`: An access modifier. `'public'` means this class can be accessed by any other class.
- `'public static void main(String[ ] args)'`: This is the **main method**.
 - The JVM starts executing the program from this method. It is the entry point of every Java application.
 - `'public'`: It can be called from anywhere.
 - `'static'`: It can be run without creating an object of the `'HelloWorld'` class.
 - `'void'`: This method does not return any value.
 - `'main'`: The name of the method.
 - `'(String[ ] args)'`: This is a parameter for the method. It can accept command-line arguments as an array of strings.
- `'System.out.println("Hello World");'`: This is the statement that does the work.
 - `'System'`: A pre-defined class in Java that provides access to the system.
 - `'out'`: An object within the `'System'` class that represents the standard output stream (usually the console).
 - `'println()'`: A method of the `'out'` object that prints the given text to the console, followed by a new line.
 - `'"Hello World"'`: The text (a String literal) to be printed.
 - `';'`: The semicolon marks the end of a statement in Java. Every statement must end with a semicolon.
- `'... '`: The curly braces define a block of code. The body of the class and the body of the method are enclosed in curly braces.

1.3 Data Types and Variables

1.3.1 Variables

A **variable** is a named container for storing data values. Think of it as a labeled box where you can put information. In Java, every variable must be declared with a specific **data type**, which determines the size and type of value it can hold.

Declaration: `'dataType variableName;'`

Example: `'int age;'`

Initialization: `'variableName = value;'`

Example: `'age = 25;'`

You can declare and initialize in a single line: `'dataType variableName = value;'`

Example: `'int age = 25;'`

1.3.2 Primitive Data Types

Java has eight primitive data types. They are called "primitive" because they are the most basic data types available in the language.

Table 1: Java Primitive Data Types

Category	Data Type	Size	Description and Example
Integers (Whole Numbers)	'byte'	8-bit	Stores whole numbers from -128 to 127. <code>'byte b = 100;'</code>
	'short'	16-bit	Stores whole numbers from -32,768 to 32,767. <code>'short s = 5000;'</code>
	'int'	32-bit	The most common integer type. <code>'int i = 100000;'</code>
	'long'	64-bit	For very large whole numbers. Value must end with 'L'. <code>'long l = 150000000000L;'</code>
Floating-Point (Decimal Numbers)	'float'	32-bit	Stores fractional numbers. Sufficient for 6-7 decimal digits. Value must end with 'f'. <code>'float f = 5.75f;'</code>
	'double'	64-bit	The most common decimal type. Sufficient for 15 decimal digits. <code>'double d = 19.99;'</code>
Character	'char'	16-bit	Stores a single character. Value is in single quotes. <code>'char grade = 'A';'</code>
Boolean	'boolean'	1-bit	Stores only two possible values: 'true' or 'false'. <code>'boolean isJavaFun = true;'</code>

1.3.3 Reference Data Types (Non-Primitive)

Unlike primitive types, reference types do not store the actual data but a reference (or memory address) to where the data is stored. The most common reference type is the **String**.

String: A 'String' is a sequence of characters. It is not a primitive type but a pre-defined class in Java. String values are enclosed in double quotes. `'String greeting = "Hello Java";'`

1.4 Operators in Java

Operators are special symbols that perform operations on variables and values (operands).

Table 2: Arithmetic Operators

Operator	Name	Example (int a=10, b=3)
'+'	Addition	'a + b' gives 13
'-'	Subtraction	'a - b' gives 7
'*'	Multiplication	'a * b' gives 30
'/'	Division	'a / b' gives 3 (integer division truncates the decimal)
'%'	Modulus	'a % b' gives 1 (returns the remainder of the division)
'++'	Increment	'a++' increases the value of 'a' by 1
'--'	Decrement	'b--' decreases the value of 'b' by 1

Table 3: Relational and Logical Operators

Operator	Name	Description
Relational Operators (return true or false)		
'=='	Equal to	Checks if two values are equal.
'!='	Not equal to	Checks if two values are not equal.
'>'	Greater than	Checks if the left value is greater than the right.
'<'	Less than	Checks if the left value is less than the right.
'>='	Greater than or equal to	Checks if the left value is greater than or equal to the right.
'<='	Less than or equal to	Checks if the left value is less than or equal to the right.
Logical Operators (combine boolean expressions)		
'&&'	Logical AND	Returns 'true' if both statements are true.
' '	Logical OR	Returns 'true' if at least one statement is true.
'!'	Logical NOT	Reverses the result; returns 'false' if the result is true, and vice versa.

1.5 Control Flow Statements

These statements control the order in which the code is executed.

1.5.1 Selection Statements

- **'if' Statement:** Executes a block of code if a specified condition is true.

```

1 if (condition) {
2     // block of code to be executed if the condition is true
3 }
4
```

- **'if-else' Statement:** Executes one block of code if the condition is true, and another block if it is false.

```

1 if (condition) {
2     // block of code to be executed if the condition is true
3 } else {
4     // block of code to be executed if the condition is false
5 }
6
```

- **'switch' Statement:** Selects one of many code blocks to be executed.

```

1 switch(expression) {
2     case x:
3         // code block
4         break;
5     case y:
6         // code block
7         break;
8     default:
9         // code block
10 }
11
```

The 'break' keyword is important. It stops the execution from continuing into the next case. The 'default' block is executed if there is no case match.

1.5.2 Repetition (Looping) Statements

- **'while' Loop:** Loops through a block of code as long as a specified condition is true. The condition is checked *before* the loop body is executed.

```

1 while (condition) {
2     // code block to be executed
3 }
4
```

- **‘do-while’ Loop:** Similar to a ‘while’ loop, but the loop body is executed at least once because the condition is checked *after* the execution.

```
1 do {
2     // code block to be executed
3 } while (condition);
4
```

- **‘for’ Loop:** Used when you know exactly how many times you want to loop through a block of code.

```
1 for (statement 1; statement 2; statement 3) {
2     // code block to be executed
3 }
4 // Statement 1: executed once before the loop starts (initialization).
5 // Statement 2: defines the condition for the loop to run.
6 // Statement 3: executed each time after the loop body has been executed (update).
7
```

1.6 Arrays

An **array** is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created.

Declaration and Creation: ‘dataType[] arrayName = new dataType[size];’

Example: ‘int[] numbers = new int[5];’ // Creates an array to hold 5 integers.

Initialization: You can initialize an array during declaration: ‘String[] cars = "Volvo", "BMW", "Ford", "Mazda";’

Accessing Elements: Array elements are accessed using their **index number**. The index starts from 0. ‘numbers[0] = 10;’ // Assigns 10 to the first element. ‘System.out.println(cars[1]);’ // Prints "BMW".

Array Length: You can find the number of elements in an array using the ‘length’ property. ‘System.out.println(cars.length);’ // Prints 4.

Looping Through an Array: A ‘for’ loop is commonly used to iterate through array elements.

```
1 for (int i = 0; i < cars.length; i++) {
2     System.out.println(cars[i]);
3 }
```

1.7 Object-Oriented Programming (OOP)

1.7.1 Class and Object

- **Class:** A class is a blueprint or a template from which objects are created. It defines the properties (attributes) and behaviors (methods) that an object of its type will have. For example, a ‘Car’ class could define properties like ‘color’ and ‘model’, and behaviors like ‘drive()’ and ‘brake()’.
- **Object:** An object is an instance of a class. It is a real-world entity that has state and behavior. For example, ‘myRedFerrari’ could be an object of the ‘Car’ class.

1.7.2 Creating a Class and Objects

```
1 public class Book {
2     // Data members (attributes)
3     String title;
4     String author;
5     int price;
6
7     // Method member (behavior)
8     public void display() {
9         System.out.println("Title: " + title);
10        System.out.println("Author: " + author);
11        System.out.println("Price: " + price);
12    }
13 }
```

Listing 2: Book.java

To use this class, we need to create objects from it in another class (usually one with a ‘main’ method).

```
1 public class BookTest {
2     public static void main(String[] args) {
3         // Create an object of the Book class
4         Book book1 = new Book();
5
6         // Access and set the data members using the dot (.) operator
7         book1.title = "The Lord of the Rings";
8         book1.author = "J.R.R. Tolkien";
9         book1.price = 500;
10
11        // Call the method
12        book1.display();
13    }
14 }
```

Listing 3: BookTest.java

1.7.3 Constructors

A **constructor** is a special method that is used to initialize objects. It is called automatically when an object is created using the ‘new’ keyword.

- A constructor has the same name as the class.
- A constructor does not have a return type (not even ‘void’).

We can create a parameterized constructor in our ‘Book’ class to initialize its data members upon creation.

```
1 public class Book {
2     String title;
3     String author;
4     int price;
5
6     // Parameterized Constructor
7     public Book(String t, String a, int p) {
8         title = t;
9         author = a;
10        price = p;
11    }
12
13    public void display() {
14        // ... same as before
15    }
16 }
```

Listing 4: Book.java with a constructor

Now, we can create an object like this: ‘Book book1 = new Book(“The Hobbit”, “J.R.R. Tolkien”, 450);’

1.7.4 Access Modifiers and Encapsulation

Encapsulation is the concept of bundling data (attributes) and methods that work on the data within a single unit (a class). It also involves restricting direct access to some of an object’s components, which is achieved using access modifiers.

- ‘public’: The member is accessible from any other class.
- ‘private’: The member is only accessible within its own class.

It is good practice to declare data members as ‘private’ to protect them from accidental modification. We then provide ‘public’ methods, called **getters** and **setters**, to access and update their values in a controlled way.

1.8 Exception Handling

An **exception** is an event that occurs during the execution of a program that disrupts the normal flow of instructions. Examples include trying to divide by zero or accessing an invalid array index.

Java's exception handling mechanism allows you to manage these runtime errors gracefully.

- **'try' block:** The code that might generate an exception is placed inside a 'try' block.
- **'catch' block:** If an exception occurs in the 'try' block, the control is transferred to the 'catch' block. The 'catch' block handles the exception.
- **'finally' block:** The 'finally' block is always executed, regardless of whether an exception occurred or not. It is often used for cleanup activities, like closing files or database connections.

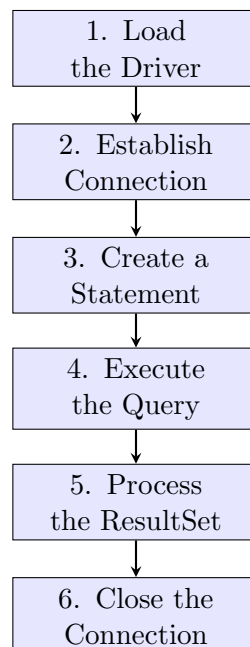
```
1 public class ExceptionExample {
2     public static void main(String[] args) {
3         try {
4             int[] myNumbers = {1, 2, 3};
5             System.out.println(myNumbers[10]); // This will cause an error
6         } catch (ArrayIndexOutOfBoundsException e) {
7             System.out.println("Error: Array index is out of bounds.");
8             System.out.println("Message from Java: " + e.getMessage());
9         } finally {
10            System.out.println("The 'try-catch' block is finished.");
11        }
12    }
13 }
```

Listing 5: Exception Handling Example

1.9 Database Connectivity (JDBC)

JDBC (Java Database Connectivity) is a Java API that allows Java programs to connect and interact with databases.

The process involves these key steps:



Here is a code skeleton demonstrating these steps.

```
1 import java.sql.*; // Import all necessary classes
2
3 public class MySQLConnect {
4     public static void main(String[] args) {
5         String url = "jdbc:mysql://localhost:3306/bookstore";
6         String username = "root";
7         String password = "password";
```

```

8      Connection con = null;
9      Statement stmt = null;
10     ResultSet rs = null;
11
12     try {
13         // 2. Establish Connection
14         con = DriverManager.getConnection(url, username, password);
15
16         // 3. Create a Statement
17         stmt = con.createStatement();
18
19         // 4. Execute the Query
20         String query = "SELECT * FROM book";
21         rs = stmt.executeQuery(query);
22
23         // 5. Process the ResultSet
24         while (rs.next()) {
25             String title = rs.getString("title");
26             String author = rs.getString("author");
27             System.out.println("Title: " + title + ", Author: " + author);
28         }
29     } catch (SQLException e) {
30         e.printStackTrace(); // Print detailed error information
31     } finally {
32         // 6. Close the Connection (in reverse order of creation)
33         try {
34             if (rs != null) rs.close();
35             if (stmt != null) stmt.close();
36             if (con != null) con.close();
37         } catch (SQLException e) {
38             e.printStackTrace();
39         }
40     }
41 }
42 }

```

Listing 6: JDBC Example

1.10 Assertions, Threads, and Wrapper Classes

1.10.1 Assertions

An **assertion** is a statement in Java that you can use to test your assumptions about the program. It checks if a condition is true, and if it is not, the program terminates by throwing an ‘AssertionError’. Assertions are primarily used for debugging during development.

There are two forms:

1. ‘assert condition;’
2. ‘assert condition : errorMessage;’

The second form allows you to provide a custom error message.

Important Note: By default, assertions are disabled at runtime. To enable them, you must use the ‘-ea’ (enable assertions) flag when running the program: ‘java -ea YourProgramName’

```

1 public class AgeValidator {
2     public static void main(String[] args) {
3         int age = 15;
4         // We assume age must be 18 or greater to proceed
5         assert age >= 18 : "Age must be at least 18.";
6         System.out.println("User is a valid adult. Age: " + age);
7     }
8 }
9 // To run this and see the error, use: java -ea AgeValidator

```

Listing 7: Assertion Example

1.10.2 Threads

A **thread** is the smallest unit of processing. A program can be **multithreaded**, meaning it can perform multiple tasks concurrently. For example, in a word processor, one thread might handle user input while another thread performs spell checking in the background.

There are two ways to create a thread in Java:

1. **By extending the ‘Thread’ class:** Create a new class that extends ‘java.lang.Thread’ and override its ‘run()’ method. The code inside the ‘run()’ method is what the thread will execute. To start the thread, you create an instance of your class and call its ‘start()’ method (not ‘run()’).

```
1 class MyThread extends Thread {
2     public void run() {
3         System.out.println("This thread is running.");
4     }
5 }
6
7 public class Main {
8     public static void main(String[] args) {
9         MyThread t1 = new MyThread();
10        t1.start(); // Starts the thread
11    }
12 }
13
```

2. **By implementing the ‘Runnable’ interface:** Create a new class that implements the ‘java.lang.Runnable’ interface and implement the ‘run()’ method. Then, create an instance of this class, pass it to the constructor of a ‘Thread’ object, and call the ‘start()’ method on the ‘Thread’ object. This is the preferred method as it allows your class to extend another class if needed.

```
1 class MyRunnable implements Runnable {
2     public void run() {
3         System.out.println("This task is running.");
4     }
5 }
6
7 public class Main {
8     public static void main(String[] args) {
9         MyRunnable task = new MyRunnable();
10        Thread t1 = new Thread(task);
11        t1.start(); // Starts the thread
12    }
13 }
14
```

1.10.3 Wrapper Classes

Java provides **wrapper classes** to allow primitive data types (like ‘int’, ‘char’, etc.) to be treated as objects. This is necessary for situations where objects are required, such as with Java’s Collection Framework (e.g., ‘ArrayList’). Each primitive type has a corresponding wrapper class.

Table 4: Primitive Types and their Wrapper Classes

Primitive Type	Wrapper Class
‘boolean’	‘Boolean’
‘byte’	‘Byte’
‘char’	‘Character’
‘short’	‘Short’
‘int’	‘Integer’
‘long’	‘Long’
‘float’	‘Float’
‘double’	‘Double’

Boxing and Unboxing:

- **Boxing** is the process of converting a primitive value into an object of the corresponding wrapper class.
- **Unboxing** is the reverse: converting an object of a wrapper type to its corresponding primitive value.

In modern Java, this conversion is often done automatically, a feature called **autoboxing** and **auto-unboxing**.

```
1 // Autoboxing: primitive int is automatically converted to an Integer object
2 Integer myIntObject = 100;
3
4 // Auto-unboxing: Integer object is automatically converted to a primitive int
5 int myIntPrimitive = myIntObject;
```

Wrapper classes also provide useful utility methods, such as ‘Integer.parseInt()’ to convert a string to an integer.

2 Practice Questions

2.1 Multiple Choice Questions (MCQs)

1. What is the output of a Java compiler?

- | | |
|------------------|-------------------|
| (A) Machine Code | (C) Source Code |
| (B) Bytecode | (D) Assembly Code |

Answer: (B)

2. Which component is responsible for converting bytecode into machine code?

- | | |
|---------|-----------|
| (A) JDK | (C) JVM |
| (B) JRE | (D) Javac |

Answer: (C)

3. Which keyword is used to define a class in Java?

- | | |
|-----------|------------|
| (A) class | (C) static |
| (B) void | (D) public |

Answer: (A)

4. What is the entry point of a Java application?

- | | |
|----------------------|---------------------|
| (A) 'start()' method | (C) 'run()' method |
| (B) 'main()' method | (D) 'init()' method |

Answer: (B)

5. Which of the following is a valid declaration of a char?

- | | |
|---------------------|-------------------------|
| (A) 'char c = "A";' | (C) 'char c = A;' |
| (B) 'char c = 'A';' | (D) 'char c = (char)A;' |

Answer: (B)

6. What is the result of '9 / 2' in Java?

- | | |
|---------|-------|
| (A) 4.5 | (C) 4 |
| (B) 5 | (D) 1 |

Answer: (C)

7. Which command-line flag is used to enable assertions when running a Java program?

- | | |
|---------------|-----------|
| (A) '-enable' | (C) '-ea' |
| (B) '-assert' | (D) '-go' |

Answer: (C)

8. Which loop will always execute its body at least once?

- (A) for
- (B) while
- (C) do-while
- (D) All of the above

Answer: (C)

9. How do you get the number of elements in an array named 'myArray'?

- (A) 'myArray.length()'
- (B) 'myArray.size'
- (C) 'myArray.length'
- (D) 'myArray.size()'

Answer: (C)

10. What is the index of the first element in a Java array?

- (A) 1
- (B) 0
- (C) -1
- (D) It depends on the array size

Answer: (B)

11. A blueprint for creating objects is called a...

- (A) Method
- (B) Variable
- (C) Class
- (D) Constructor

Answer: (C)

12. Which method must be called to start the execution of a new thread?

- (A) 'run()'
- (B) 'new()'
- (C) 'start()'
- (D) 'execute()'

Answer: (C)

13. A method that is used to initialize an object is called a...

- (A) Method
- (B) Variable
- (C) Class
- (D) Constructor

Answer: (D)

14. Which access modifier makes a member accessible only within its own class?

- (A) 'public'
- (B) 'private'
- (C) 'protected'
- (D) 'default'

Answer: (B)

15. Which block is used to handle exceptions?

- (A) 'try'
- (B) 'catch'
- (C) 'finally'
- (D) 'throw'

Answer: (B)

16. What is the primary purpose of wrapper classes?

- (A) To make code run faster
- (B) To treat primitive types as objects
- (C) To secure primitive data
- (D) To replace primitive types

Answer: (B)

17. What does JDBC stand for?

- (A) Java Database Connection
- (B) Java Data Binding Class
- (C) Java Database Connectivity
- (D) Java Data Class

Answer: (C)

18. Which method is used to execute a 'SELECT' query in JDBC?

- (A) 'executeUpdate()'
- (B) 'execute()'
- (C) 'executeQuery()'
- (D) 'runQuery()'

Answer: (C)

19. The result of a 'SELECT' query is returned in an object of which class?

- (A) 'Statement'
- (B) 'Connection'
- (C) 'DataTable'
- (D) 'ResultSet'

Answer: (D)

20. Which character marks the end of a Java statement?

- (A) '.' (Period)
- (B) ',' (Comma)
- (C) ';' (Semicolon)
- (D) ':' (Colon)

Answer: (C)

2.2 One-Word Answer Questions

1. The platform-independent intermediate code generated by the Java compiler. (Bytecode)
2. The full form of JVM. (Java Virtual Machine)
3. The full form of JDK. (Java Development Kit)
4. The keyword used to indicate that a method does not return a value. (void)
5. The data type for storing true or false values. (boolean)
6. The operator used to find the remainder of a division. (%)
7. The logical AND operator in Java. (&&)
8. The statement used to terminate a 'switch' case. (break)
9. A named memory location for storing data. (Variable)
10. The process of creating an object from a class. (Instantiation)
11. The error thrown when an assertion fails. (AssertionError)
12. An instance of a class. (Object)
13. The operator used to access members of an object. (.)

14. The concept of bundling data and methods together in a class. (Encapsulation)
15. The interface you can implement to create a thread. (Runnable)
16. A runtime error that disrupts the normal flow of a program. (Exception)
17. The block where code that might throw an exception is placed. (try)
18. The API for Java database connectivity. (JDBC)
19. The wrapper class for the 'int' primitive type. (Integer)
20. The file extension for a Java source code file. (.java)

2.3 Differentiate Between Questions

1. Compiler vs. Interpreter

Answer: A compiler translates the entire source code into machine code at once, creating a separate executable file. An interpreter translates and executes the source code line by line, without creating a separate executable file.

2. 'while' loop vs. 'do-while' loop

Answer: A 'while' loop is an entry-controlled loop where the condition is checked before the loop body is executed. It may not execute at all if the condition is initially false. A 'do-while' loop is an exit-controlled loop where the condition is checked after the loop body is executed, ensuring the body is executed at least once.

3. Primitive Data Types vs. Wrapper Classes

Answer: Primitive data types (like 'int', 'char') store the actual data value directly in memory. Wrapper classes ('Integer', 'Character') are objects that "wrap" a primitive value, allowing it to be treated as an object, which is necessary for use in collections and other object-oriented scenarios.

4. Class vs. Object

Answer: A class is a blueprint or template that defines the properties and behaviors for a type of entity. An object is a concrete instance of a class, with its own specific values for the properties defined in the class.

5. Extending 'Thread' vs. Implementing 'Runnable'

Answer: Both are ways to create a thread. Extending the 'Thread' class is simpler but means your class cannot extend any other class due to Java's single inheritance rule. Implementing the 'Runnable' interface is more flexible as it allows your class to extend another class while still being able to run as a thread.

6. 'start()' vs. 'run()' method for threads

Answer: The 'start()' method is used to begin the execution of a new thread. It registers the thread with the thread scheduler and then calls the 'run()' method. Calling the 'run()' method directly does not start a new thread; it simply executes the code in the 'run()' method in the current thread.

7. Method vs. Constructor

Answer: A method is a block of code that performs a task and can have any name and a return type. A constructor is a special method used to initialize an object, must have the same name as the class, and has no return type.

8. 'private' vs. 'public' access modifiers

Answer: A 'public' member (variable or method) can be accessed from any other class. A 'private' member can only be accessed from within its own class, enforcing encapsulation.

9. 'executeQuery()' vs. 'executeUpdate()' in JDBC

Answer: 'executeQuery()' is used for SQL statements that retrieve data, such as 'SELECT'. It returns a 'ResultSet' object. 'executeUpdate()' is used for SQL statements that modify data, such as 'INSERT', 'UPDATE', or 'DELETE'. It returns an integer representing the number of rows affected.

10. JDK vs. JRE

Answer: JRE (Java Runtime Environment) provides the necessary components (JVM and libraries) to run Java applications. JDK (Java Development Kit) includes everything in the JRE plus development tools (like the compiler and debugger) needed to create Java applications.

2.4 Short Answer Questions

1. What is the Java Virtual Machine (JVM) and why is it important?

Answer: The JVM is an abstract machine that provides a runtime environment in which Java bytecode can be executed. It is important because it is the component that makes Java platform-independent; it translates the universal bytecode into native machine code for the specific operating system it is running on.

2. Explain the purpose of a constructor in Java.

Answer: A constructor is a special method used to initialize a newly created object. It is called automatically when an object is instantiated using the 'new' keyword and is primarily used to set initial values for the object's data members.

3. Why are wrapper classes necessary in Java?

Answer: Wrapper classes are necessary because many parts of the Java API, especially the Collection Framework (like 'ArrayList'), can only work with objects. Wrapper classes allow primitive data types like 'int' or 'double' to be "wrapped" into an object, so they can be used in these object-only contexts.

4. What is an array in Java? Give one advantage and one disadvantage.

Answer: An array is a data structure that stores a fixed-size collection of elements of the same data type. An advantage is fast access to elements using an index. A disadvantage is its fixed size; once created, the size of an array cannot be changed.

5. Explain the two ways to create a thread in Java.

Answer: The two ways are by extending the 'Thread' class and overriding its 'run()' method, or by implementing the 'Runnable' interface and implementing the 'run()' method. The second approach is generally preferred as it allows the class to inherit from another class if needed.

6. What is an assertion and when should it be used?

Answer: An assertion is a statement used to check for bugs by verifying assumptions made by the programmer. It should be used during development and testing to catch logical errors early, but should be disabled in production code as it can impact performance.

7. What is the purpose of the 'break' statement inside a 'switch' block?

Answer: The 'break' statement is used to terminate the 'switch' block. Without 'break', the program would continue to execute the code in the subsequent 'case' blocks (a behavior known as "fall-through") until a 'break' or the end of the 'switch' is reached.

8. What is encapsulation and how is it achieved in Java?

Answer: Encapsulation is an OOP principle of bundling the data (attributes) and the methods that operate on the data into a single unit, or class. It is achieved in Java by declaring the data members as 'private' and providing 'public' getter and setter methods to control access to them.

9. What is the role of the 'ResultSet' object in JDBC?

Answer: The 'ResultSet' object is used to hold the data retrieved from a database after executing a 'SELECT' query. It maintains a cursor pointing to its current row of data, and methods like 'next()' are used to move the cursor through the rows to process the results.

10. What is the purpose of the 'import' statement in Java?

Answer: The 'import' statement is used to bring classes or entire packages into the current scope, allowing you to use them in your code without specifying their full package name. For example, 'import java.util.Scanner;' lets you use the 'Scanner' class directly.

2.5 Long Answer Questions

1. Explain the entire process of Java code compilation and execution, from a '.java' file to the final output. Mention the key components involved and draw a simple diagram to illustrate the flow.

Answer: The Java compilation and execution process is a two-step hybrid model that ensures platform independence.

- (a) **Compilation:** First, a programmer writes the source code in a file with a '.java' extension. This code is then fed to the Java compiler, 'javac'. The compiler checks the code for syntax errors and, if it's correct, translates it not into machine code, but into an intermediate, platform-independent code called Java Bytecode. This bytecode is stored in a '.class' file.
- (b) **Execution (Interpretation):** The generated '.class' file can be run on any machine that has a Java Virtual Machine (JVM). When the program is executed (e.g., using the 'java' command), the JVM loads the '.class' file. The JVM then acts as an interpreter; it reads the bytecode line by line, translates it into the native machine code specific to the underlying operating system and hardware, and then executes it. This "just-in-time" translation by the JVM is what allows a single Java '.class' file to run anywhere.

The key components are the Java Compiler ('javac') which creates bytecode, and the Java Virtual Machine (JVM) which interprets the bytecode to run the program. (A diagram similar to the one in the notes section would be appropriate here).

2. Describe the main principles of Object-Oriented Programming (OOP) with simple examples related to a 'Student' class.

Answer: Object-Oriented Programming is a paradigm based on the concept of "objects". The main principles are:

- **Encapsulation:** This is the bundling of data (attributes) and methods (behaviors) that operate on the data into a single unit, the class. It also involves data hiding by making attributes 'private' and providing 'public' methods (getters/setters) for access. For a 'Student' class, attributes like 'rollNumber' and 'name' would be 'private', and we would have public methods like 'getRollNumber()' and 'setName()'.
- **Abstraction:** This means hiding complex implementation details and showing only the essential features of the object. For example, when we call a 'calculateGrade()' method on a 'Student' object, we don't need to know the complex logic inside it; we only need to know that it will return the student's grade.
- **Inheritance:** This is a mechanism where a new class (subclass) can inherit attributes and methods from an existing class (superclass). This promotes code reusability. For example, a 'GraduateStudent' class could inherit from the 'Student' class and add its own specific attributes like 'thesisTopic'.
- **Polymorphism:** This means "many forms". It allows an object to take on many forms. For example, a method 'displayDetails()' could behave differently for a 'Student' object versus a 'GraduateStudent' object, even though they are related through inheritance.

3. Write a complete Java program to declare an integer array, initialize it with at least five values, and then find and print the largest element in the array.

Answer:

```
1 public class FindLargestElement {
2     public static void main(String[] args) {
3         // 1. Declare and initialize an integer array
4         int[] numbers = {25, 14, 56, 15, 36, 89, 77};
5
6         // 2. Assume the first element is the largest initially
7         int largest = numbers[0];
8
9         // 3. Loop through the array starting from the second element
10        for (int i = 1; i < numbers.length; i++) {
11            // 4. Compare the current element with the current largest
12            if (numbers[i] > largest) {
13                // If the current element is larger, update largest
```

```

14         largest = numbers[i];
15     }
16 }
17
18 // 5. Print the result
19 System.out.println("The largest element in the array is: " + largest);
20 }
21 }
22

```

4. Explain the steps and key classes/interfaces involved in connecting a Java application to a MySQL database using JDBC. Provide a code structure with comments explaining each part.

Answer: Connecting a Java application to a MySQL database using JDBC involves several steps and key classes from the 'java.sql' package.

- Load the Driver:** Although modern JDBC versions often handle this automatically, the first step is to load the MySQL JDBC driver.
- Establish Connection:** Use the 'DriverManager.getConnection()' method. This requires a database URL, username, and password. The URL for MySQL has the format 'jdbc:mysql://hostname:port/database'. This method returns a 'Connection' object.
- Create a Statement:** From the 'Connection' object, create a 'Statement' object using the 'createStatement()' method. The 'Statement' object is used to send SQL queries to the database.
- Execute the Query:** Use the 'Statement' object's 'executeQuery()' method for 'SELECT' queries or 'executeUpdate()' for 'INSERT', 'UPDATE', 'DELETE' queries. 'executeQuery()' returns a 'ResultSet' object.
- Process Results:** If a 'ResultSet' is returned, iterate through it using a 'while(rs.next())' loop to access the data from each row.
- Close Connections:** It is crucial to close the 'ResultSet', 'Statement', and 'Connection' objects in a 'finally' block to release database resources.

(The code structure would be the same as the JDBC example in the notes).

5. Discuss the three main types of loops in Java ('for', 'while', 'do-while'). Explain their syntax, control flow (entry-controlled vs. exit-controlled), and provide a scenario where each would be the most appropriate choice.

Answer:

- **'for' loop:**

- **Syntax:** 'for (initialization; condition; update) // body '
- **Control Flow:** It is an entry-controlled loop. The initialization runs once, then the condition is checked. If true, the body executes, followed by the update statement. This repeats until the condition is false.
- **Best Scenario:** Use a 'for' loop when the number of iterations is known beforehand, such as iterating through all elements of an array or looping a fixed number of times.

- **'while' loop:**

- **Syntax:** 'while (condition) // body '
- **Control Flow:** It is an entry-controlled loop. The condition is checked before each iteration. If the condition is true, the body executes. If it's false initially, the loop body never executes.
- **Best Scenario:** Use a 'while' loop when the number of iterations is not known in advance, and the loop should continue as long as a certain condition holds true, for example, reading from a file until the end is reached.

- **'do-while' loop:**

- **Syntax:** 'do // body while (condition);'
- **Control Flow:** It is an exit-controlled loop. The loop body is executed first, and then the condition is checked. This guarantees that the loop body is executed at least once.

- **Best Scenario:** Use a ‘do-while’ loop when the loop must execute at least one time, regardless of the condition. A common example is a menu-driven program that displays options and asks for user input, which must happen at least once.

2.6 Practical Application Questions

1. Write the step-by-step code to take a user’s name (as a String) and age (as an integer) as input and then print a message like ”Hello [Name], you are [Age] years old.”

Answer:

```
1 // Step 1: Import the Scanner class
2 import java.util.Scanner;
3
4 public class UserInput {
5     public static void main(String[ ] args) {
6         // Step 2: Create a Scanner object to read input from the console
7         Scanner input = new Scanner(System.in);
8
9         // Step 3: Prompt the user for their name and read it
10        System.out.print("Please enter your name: ");
11        String name = input.nextLine();
12
13        // Step 4: Prompt the user for their age and read it
14        System.out.print("Please enter your age: ");
15        int age = input.nextInt();
16
17        // Step 5: Print the formatted output message
18        System.out.println("Hello " + name + ", you are " + age + " years old.");
19
20        // Step 6: Close the scanner to release resources
21        input.close();
22    }
23 }
24
```

2. Describe the steps to create a new Java class named 'Student' with private data members for 'rollNo' (int) and 'name' (String), and include public getter and setter methods for both.

Answer:

```
1 public class Student {
2     // Step 1: Declare private data members
3     private int rollNo;
4     private String name;
5
6     // Step 2: Create a public getter method for rollNo
7     public int getRollNo() {
8         return rollNo;
9     }
10
11    // Step 3: Create a public setter method for rollNo
12    public void setRollNo(int newRollNo) {
13        this.rollNo = newRollNo; // 'this' keyword refers to the current object
14    }
15
16    // Step 4: Create a public getter method for name
17    public String getName() {
18        return name;
19    }
20
21    // Step 5: Create a public setter method for name
22    public void setName(String newName) {
23        this.name = newName;
24    }
25 }
26
```

3. Provide the step-by-step code to create a String array of your friends' names, sort it alphabetically, and then print the sorted list.

Answer:

```
1 // Step 1: Import the Arrays class for sorting functionality
2 import java.util.Arrays;
3
4 public class SortFriends {
5     public static void main(String[] args) {
6         // Step 2: Create and initialize a String array
7         String[] friends = {"Rohan", "Priya", "Amit", "Zoya", "Bhavin"};
8
9         // Step 3: Sort the array alphabetically using Arrays.sort()
10        Arrays.sort(friends);
11
12        // Step 4: Print the sorted list using a for loop
13        System.out.println("Sorted list of friends:");
14        for (int i = 0; i < friends.length; i++) {
15            System.out.println(friends[i]);
16        }
17    }
18 }
19
```

4. Outline the complete code structure required to fetch all records from a 'products' table (with columns 'product_name' and 'price') in a MySQL database named 'inventory'.

Answer:

```
1 import java.sql.*;
2
3 public class FetchProducts {
4     public static void main(String[] args) {
5         // Database credentials and URL
6         String url = "jdbc:mysql://localhost:3306/inventory";
7         String user = "root";
8         String pass = "your_password";
9
10        // SQL query
11        String query = "SELECT product_name, price FROM products";
12
13        // Use try-with-resources to automatically close connections
14        try (Connection con = DriverManager.getConnection(url, user, pass);
15             Statement stmt = con.createStatement();
16             ResultSet rs = stmt.executeQuery(query)) {
17
18            System.out.println("--- Product List ---");
19            // Loop through the result set
20            while (rs.next()) {
21                String productName = rs.getString("product_name");
22                double price = rs.getDouble("price");
23                System.out.println("Product: " + productName + ", Price: " + price);
24            }
25
26        } catch (SQLException e) {
27            System.out.println("Database connection failed!");
28            e.printStackTrace();
29        }
30    }
31 }
32
```

5. Write a Java program that demonstrates a 'do-while' loop by repeatedly asking the user to enter a number until they enter the number 0.

Answer:

```
1 import java.util.Scanner;
2
3 public class GuessUntilZero {
4     public static void main(String[] args) {
```

```

5 Scanner input = new Scanner(System.in);
6 int number; // Declare the variable outside the loop
7
8 // Use a do-while loop to ensure the block runs at least once
9 do {
10     // Prompt the user for input
11     System.out.print("Enter a number (enter 0 to quit): ");
12     number = input.nextInt();
13
14     // Provide feedback
15     if (number != 0) {
16         System.out.println("You entered: " + number);
17     }
18
19 } while (number != 0); // The loop continues as long as the number is not 0
20
21 System.out.println("You entered 0. Program terminated.");
22 input.close();
23 }
24 }
25

```