JAVA UNIT 1

1. Define Tokens in Java

- Tokens are the **smallest meaningful units** in a Java program. Java compiler reads a program by breaking it into tokens.
- Tokens help the compiler understand **syntax and structure** of the program. Every Java statement is made using multiple tokens. Tokens improve program readability and proper execution.
- Java programs consist of the following tokens:
    - **Keywords**
    - **Identifiers**
    - **Literals**
    - **Operators**
    - **Separators**
    - **Comments**
    - **Whitespace**
- Whitespace separates tokens but has no effect on execution. Comments are ignored by the compiler.
- Example:
  int a = 10;
  Tokens are: int, a, =, 10, ;
- Without correct tokens, the program will generate **compile-time errors**.

2. Define Keywords and List Java Keywords

- Keywords are **reserved words** in Java. Each keyword has a **predefined meaning**.
- Keywords are used to define:
    - Data types
    - Control statements
    - Class structure
    - Access control
- Keywords **cannot be used** as variable, method, or class names. Java has **50 keywords**.
- All keywords are written in **lowercase letters**.

- Some commonly used Java keywords are:
    - int, double, char, boolean
    - if, else, switch, case
    - for, while, do, break, continue
    - class, public, private, protected
    - static, final, void, return
    - new, this
- Java also has reserved but unused keywords:
    - const
    - goto
- Using keywords incorrectly causes **compilation errors**.

3. Explain Rules for Naming Identifiers in Java

- Identifiers are names given to:
    - Variables , Methods And Classes .
- An identifier can contain:
    - Uppercase letters (A–Z)
    - Lowercase letters (a–z)
    - Digits (0–9)
    - Underscore (_)
    - Dollar sign ($)
- An identifier **must not start with a number**. Java identifiers are **case-sensitive**.
    - Value and value are different.
- Keywords cannot be used as identifiers.
- Special characters like @, #, -, / are not allowed.
- Valid identifiers:
    - count, AvgTemp, a4, this_is_ok
- Invalid identifiers:
    - 2count, high-temp, Not/ok
- Following naming rules avoids errors and improves readability.

4. Explain Different Data Types in Java with Examples

- Data types specify **what type of value** a variable can store. Java mainly uses **primitive data types**. Primitive data types store **single values**.
- Java has **8 primitive data types**:
- **Integer types**:
    - byte (8 bits)
    - short (16 bits)
    - int (32 bits)
    - long (64 bits)
- **Floating-point types**:
    - float (32 bits)
    - double (64 bits)
- **Character type**:
    - char (16 bits, Unicode)
- **Boolean type**:
    - boolean (true or false)
- Examples:
    - int age = 20;
    - double price = 99.99;
    - char grade = 'A';
    - boolean result = true;
- Correct data type selection saves memory and improves performance.

5. What is a Variable? Write Syntax for Variable Declaration

- A variable is a **basic unit of storage** in Java. It is used to store values during program execution.
- A variable has:
    - Data type
    - Identifier (name)
    - Optional initial value
- Variables allow data to **change during runtime**.
- Variables must be declared **before use**.

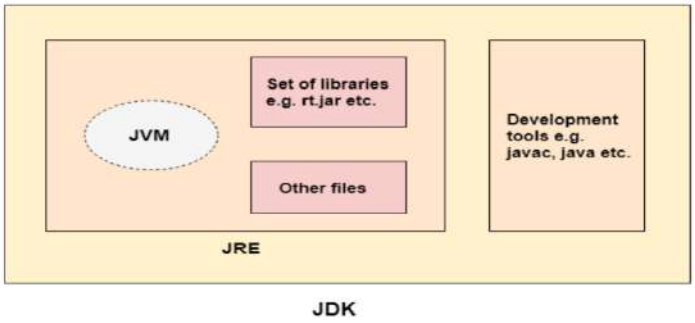
- General syntax for variable declaration:
    - type identifier = value;
- Examples:
    - int number = 10;
    - double salary = 45000.5;
    - char grade = 'B';
    - boolean isActive = true;
- Java also supports **dynamic initialization**:
    - double c = Math.sqrt(a * a + b * b);
- Variables make programs **flexible and reusable**.

6. Explain Features of Java

- **Simple:**
  Java is easy to learn and understand, especially for programmers who know basic OOP concepts. It avoids complex features like pointers and multiple inheritance.
- **Object-Oriented:**
  Java follows object-oriented principles such as encapsulation, inheritance, abstraction, and polymorphism. Everything in Java is written inside classes and objects.
- **Platform Independent:**
  Java programs are compiled into bytecode, which can run on any system with a JVM. This supports the concept of “Write Once, Run Anywhere”.
- **Robust:**
  Java has strong memory management, exception handling, and type checking. These features help prevent runtime errors and crashes.
- **Secure:**
  Java does not use pointers and runs programs inside the JVM environment. This prevents unauthorized access to system resources.
- **Multithreaded:**
  Java supports multithreading, allowing multiple tasks to run simultaneously. This improves CPU utilization and performance.

7. Explain JDK, JRE, and JVM with Diagram, Relationship, and Components of JDK

- **JVM (Java Virtual Machine):**
  JVM is responsible for executing Java bytecode. Each operating system has its own JVM, but the output remains the same.
- **JRE (Java Runtime Environment):**
  JRE provides the runtime environment required to run Java programs. It contains JVM and core libraries.
- **JDK (Java Development Kit):**
  JDK is a complete package used to develop Java programs. It includes JRE, compiler, debugger, and development tools.



- **Components of JDK:**
  JDK includes the Java compiler (javac), JVM, JRE, debugging tools, and documentation tools. These components help in writing, compiling, and running Java programs.

8. Define Type Conversion and Explain Different Ways

- **Type Conversion Definition:**
  Type conversion is the process of converting one data type into another. It allows compatibility between different data types.
- **Implicit Type Conversion (Widening):**
  This conversion happens automatically when a smaller data type is converted into a larger one. No data loss occurs in this process.
- **Rules for Implicit Conversion:**
  The source and destination data types must be compatible. The destination type must be larger than the source type.
- **Explicit Type Conversion (Narrowing):**
  This conversion is done manually by the programmer. It is required when converting a larger data type into a smaller one.
- **Need for Type Conversion:**
  Type conversion helps in performing calculations between different data types. It ensures correct execution of expressions.

9. Define Type Casting and Explain Its Types

- **Type Casting Definition:**
  Type casting is a form of explicit type conversion. It is used when automatic conversion is not possible.
- **Widening Type Casting:**
  In this type, a smaller data type is converted into a larger data type. This casting is safe and happens automatically.
- **Narrowing Type Casting:**
  In this type, a larger data type is converted into a smaller one. Data loss may occur, so explicit casting is required.
- **Syntax of Type Casting:**
  The syntax is (target_type) value. The programmer must specify the desired data type.
- **Example:**
  Converting double to int removes the decimal part. This is commonly used when precision is not required.

10. Explain if-else Statement with Example Program

- **Definition:**
  The if-else statement is a decision-making statement in Java. It executes different blocks of code based on a condition.
- **if Block:**
  The if block executes when the condition is true. The condition must return a boolean value.
- **else Block:**
  The else block executes when the condition is false. It is optional but useful for alternative execution.
- **Working:**
  Java checks the condition first. Based on the result, only one block is executed.
- **Example Program:**

```
public class IfElseExample {
  public static void main(String[] args) {
    int number = -5;
    if (number > 0) {
      System.out.println("Number is positive");
    } else {
      System.out.println("Number is negative or zero");
    }
  }}
```

11. Explain switch-case with Example Program

- The **switch-case statement** is a selection control statement in Java. It allows execution of different blocks of code based on the value of an expression.
- It is generally used as an alternative to long **if-else-if ladders**. This makes the program more readable and efficient.
- The expression in switch must be of type **byte, short, int, char, or String**. Each case value must be a constant.
- The **break statement** is used to stop execution after a matching case. If break is not used, execution continues to the next case.
- The **default case** executes when no case matches the expression. It is optional but recommended.

Example Program:

```
class SwitchExample {
  public static void main(String[] args) {
    int day = 3;
    switch(day) {
      case 1: System.out.println("Monday"); break;
      case 2: System.out.println("Tuesday"); break;
      case 3: System.out.println("Wednesday"); break;
      default: System.out.println("Invalid day");
    }
  }
}
```

12. Explain while Loop with Purpose, Syntax, and Example

- The **while loop** is an iteration statement used to execute a block of code repeatedly. It runs as long as the condition remains true.
- It is called an **entry-controlled loop** because the condition is checked before executing the loop body.
- The main purpose of the while loop is to repeat tasks when the number of iterations is not known in advance. It is commonly used for reading input or looping until a condition changes.
- If the condition becomes false, the loop stops immediately.

Syntax:

```
while(condition) {
  // loop body
}
```

Example Program:

```
class WhileExample {
  public static void main(String[] args) {
    int i = 1;
    while(i <= 5) {
      System.out.println(i);
      i++;
    }
  }}
```

13. Explain for Loop with Purpose, Syntax, and Example

- The **for loop** is used when the number of iterations is known in advance. It combines initialization, condition, and increment in one line.
- It improves readability and reduces code length. This makes it ideal for counting loops.
- The loop executes as long as the condition is true. After each iteration, the update statement is executed.
- The for loop is commonly used with arrays and counters.

Syntax:

```
for(initialization; condition; increment/decrement) {
  // loop body
}
```

Example Program:

```
class ForExample {
  public static void main(String[] args) {
    for(int i = 1; i <= 5; i++) {
      System.out.println(i);
    }
  }
}
```

14. Explain do-while Loop with Example Program

- The **do-while loop** is an iteration statement that executes the loop body at least once. The condition is checked after execution.
- It is called an **exit-controlled loop** because the condition is evaluated at the end.
- This loop is useful when the loop body must run at least once, regardless of the condition.
- Even if the condition is false initially, the loop executes once.

Syntax:

```
do {
  // loop body
} while(condition);
```

Example Program:

```
class DoWhileExample {
  public static void main(String[] args) {
    int i = 1;
    do {
      System.out.println(i);
      i++;
    } while(i <= 5);
  }
}
```

15. Explain for-each Loop with Array Sum Program

- The **for-each loop** is used to traverse arrays and collections. It accesses elements directly instead of using index values.
- It is simpler and cleaner than a traditional for loop. This reduces chances of index-related errors.
- The for-each loop is **read-only**, meaning it cannot modify the original array elements.
- It is mainly used for processing each element in an array.

Syntax:

```
for(dataType variable : array) {
  // loop body
}
```

Array Sum Program:

```
class ArraySum {
  public static void main(String[] args) {
    int[] nums = {10, 20, 30, 40};
    int sum = 0;
    for(int n : nums) {
      sum += n;
    }
    System.out.println("Sum = " + sum);
  }
}
```

16. Explain break Statement with Purpose, Syntax, and Example

- The **break statement** is a jump control statement in Java. It is used to immediately stop the execution of a loop or switch statement.
- When the break statement is executed, program control moves **outside the loop or switch block**. No further iterations are performed after break.
- The main purpose of break is to **terminate a loop early** when a desired condition is satisfied. This helps improve performance by avoiding unnecessary iterations.

Syntax:

```
break;
```

Program to Search an Element Using break:

```
class SearchArray {
  public static void main(String[] args) {
    int[] arr = {10, 20, 30, 40, 50};
    int key = 30;
    for(int i = 0; i < arr.length; i++) {
      if(arr[i] == key) {
        System.out.println("Element found at index " + i);
        break;
      }
    }
  }}
```

17. Explain continue Statement with Skipping Multiples Program

- The **continue statement** is a loop control statement used to skip the remaining statements of the current iteration. After continue, the loop proceeds with the next iteration.
- Unlike break, continue does **not terminate the loop completely**. It only skips the current loop cycle.
- The main purpose of continue is to **ignore specific values or conditions** while looping. This helps write cleaner and more readable logic.
- Continue is useful when some values should not be processed, such as skipping even numbers or multiples.
- It can be used inside **for, while, and do-while loops**.
- In a for loop, continue transfers control to the increment statement. In while and do-while loops, control moves to the condition check.

Syntax:

```
continue;
```

Program to Skip Multiples of 5:

```
class ContinueExample {
  public static void main(String[] args) {
    for(int i = 1; i <= 20; i++) {
      if(i % 5 == 0) {
        continue;
      }
      System.out.println(i);
    }
  }
}
```

18. What is Array? Explain Advantages and Disadvantages

- An **array** is a collection of elements of the **same data type** stored in contiguous memory locations. Each element is accessed using an index.
- Arrays allow storing multiple values under a **single variable name**, which simplifies program structure.
- The index of an array starts from **0** in Java. This helps access elements efficiently.

Advantages of Arrays

- Arrays provide **fast access** to elements using index values. This improves program efficiency.
- They help organize large amounts of data in a structured manner. Code becomes easier to manage.
- Arrays reduce memory overhead compared to using multiple individual variables.

Disadvantages of Arrays

- The size of an array is **fixed** and cannot be changed at runtime. This may cause memory wastage.
- Arrays store only **homogeneous data types**. Different data types cannot be stored together.
- Insertion and deletion operations are difficult. Shifting of elements is required.

19. Explain 1D and 2D Arrays with Syntax and Example

- A **one-dimensional (1D) array** stores data in a linear form. It is similar to a list of values.
- A **two-dimensional (2D) array** stores data in rows and columns. It is useful for tables and matrices.
- 1D arrays are simple and easy to use. They are commonly used for storing marks, numbers, or names.
- 2D arrays are more complex and require nested loops for processing. They are used in matrix operations.

1D Array Syntax & Example:

```
int[] a = new int[5];

a[0] = 10;
```

2D Array Syntax & Example:

```
int[][] b = new int[2][2];

b[0][0] = 1;

b[0][1] = 2;
```

- Both arrays store data of the same type. The main difference lies in their structure.

20. WAP to Add Two Dimensional Matrices Using 2D Array

- Matrix addition means adding corresponding elements of two matrices. Both matrices must have the **same order**.
- A **2D array** is ideal for storing matrix elements in Java. Rows and columns are represented using nested arrays.
- Two nested loops are used to traverse rows and columns. Each element is added and stored in a third matrix.
- Matrix addition is widely used in scientific and mathematical applications.
- The result matrix stores the sum of corresponding elements.

Java Program:

```
class MatrixAddition {
   public static void main(String[] args) {
      int[][] a = {{1, 2}, {3, 4}};
      int[][] b = {{5, 6}, {7, 8}};
      int[][] sum = new int[2][2];

      for(int i = 0; i < 2; i++) {
         for(int j = 0; j < 2; j++) {
            sum[i][j] = a[i][j] + b[i][j];
            System.out.print(sum[i][j] + " ");
         }
         System.out.println();
      }
   }
}
```

- This program demonstrates proper use of 2D arrays. Nested loops ensure correct traversal and addition.

21. Explain How to Read Input from the User in Java

- Java allows user input using classes like **Scanner** and **BufferedReader**. These classes read data from the keyboard.
- The most commonly used class is **Scanner**, which belongs to the java.util package. It is easy to use and beginner-friendly.
- To read input, we create a Scanner object using System.in. This connects the program to keyboard input.
- Scanner provides different methods to read different data types. This includes integers, floating-point numbers, characters, and strings.
- Some commonly used methods are nextInt(), nextDouble(), nextFloat(), next(), and nextLine().
- Scanner does not automatically consume newline characters. This can cause issues when switching between numeric input and string input.
- Reading user input makes programs interactive. It allows users to enter values during runtime.

Program to Read Different Data Types:

```
import java.util.Scanner;

class UserInput {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter integer: ");
      int i = sc.nextInt();
      System.out.print("Enter double: ");
      double d = sc.nextDouble();
      System.out.print("Enter character: ");
      char c = sc.next().charAt(0);
      System.out.print("Enter string: ");
      sc.nextLine();
      String s = sc.nextLine();
      System.out.println(i + " " + d + " " + c + " " + s);
   }
}
```

22. Explain String Class and Its Methods

- The **String class** in Java is used to represent a sequence of characters. It belongs to the java.lang package.
- Strings in Java are **immutable**, meaning their values cannot be changed once created. Any modification creates a new object.
- String objects can be created using string literals or by using the new keyword.
- The immutability of String makes Java more secure and memory efficient. It is widely used in applications.
- The String class provides many built-in methods for string manipulation.
- Common methods include length(), charAt(), equals(), compareTo(), toUpperCase(), and toLowerCase().
- Methods like concat() and substring() are used to combine and extract strings.
- String comparison should be done using equals() and not ==. This ensures content comparison.

23. Explain StringBuffer Class and Its Methods

- **StringBuffer** is a class used to create mutable (modifiable) strings. It belongs to the java.lang package.
- Unlike String, StringBuffer allows modification without creating new objects. This improves performance.
- StringBuffer is **thread-safe**, meaning it is safe to use in multi-threaded environments.
- It is preferred when frequent changes to strings are required.
- StringBuffer uses a dynamic buffer to store characters.
- Common methods include append(), insert(), delete(), reverse(), and replace().
- The append() method adds text at the end of the string. This is faster than string concatenation.
- The reverse() method reverses the string content.
- StringBuffer is useful in loops and large text processing tasks.

24. Differentiate Between String and StringBuffer with Program

Differences

- **String** objects are immutable, while **StringBuffer** objects are mutable.
- Any modification to a String creates a new object. StringBuffer modifies the same object.
- String is faster for fixed data, but StringBuffer is better for frequent changes.
- StringBuffer is thread-safe, while String is not synchronized.

Program Illustration:

```
class StringVsBuffer {
   public static void main(String[] args) {
      String s = "Hello";
      s = s + " World";
      System.out.println(s);

      StringBuffer sb = new StringBuffer("Hello");
      sb.append(" World");
      System.out.println(sb);
   }
}
```

25. WAP to Reverse a String Without Using Inbuilt Methods

- Reversing a string means changing the order of characters.
- This can be done using a loop and character indexing.
- No inbuilt methods like reverse() are used.
- A loop starts from the last character and prints characters one by one.
- This logic improves understanding of strings and loops.
- Manual reversal is useful for exams and logic building.

Program:

```
class ReverseString {
   public static void main(String[] args) {
      String s = "JAVA";
      for(int i = s.length() - 1; i >= 0; i--) {
         System.out.print(s.charAt(i));
      }
   }
}
```

26. Compare Scanner and BufferedReader Classes

- **Scanner is easy to use and supports multiple data types directly.**
  Scanner provides built-in methods like nextInt(), nextDouble(), and nextLine() to read different data types. This makes it very convenient for beginners and small programs.
- **BufferedReader is faster and suitable for large input.**
  BufferedReader reads data in large chunks, which improves performance. It is commonly used when handling large input files or competitive programming problems.
- **Scanner is slower because it performs parsing internally.**
  Scanner does additional work like breaking input into tokens and converting data types. This internal parsing makes it slower compared to BufferedReader.
- **BufferedReader reads input as text and needs conversion.**
  BufferedReader reads input only in string format using readLine(). The programmer must manually convert the string into required data types.
- **Scanner belongs to the java.util package.**
  This package contains utility classes that are simple and user-friendly. Scanner is designed mainly for ease of input handling.
- **BufferedReader belongs to the java.io package.**
  The java.io package is meant for input-output operations. BufferedReader is used for efficient reading of characters and text.
- **Scanner is preferred for small programs and beginners.**
  Its syntax is simple and easy to remember. It reduces coding complexity for basic input tasks.
- **BufferedReader is preferred in competitive programming and large data input.**
  It provides better performance and faster execution. This makes it suitable for time-critical applications.

27. Explain Platform Independence of Java

- **Platform independence means a program can run on any operating system.**
  A Java program written on one system can run on another system without modification. This is one of the most important features of Java.
- **Java achieves this using bytecode and JVM.**
  Java programs are not directly converted into machine-specific code. Instead, they are converted into an intermediate form called bytecode.
- **Java source code is compiled into bytecode by the Java compiler.**
  The Java compiler (javac) converts .java files into .class files. These class files contain bytecode.
- **Bytecode is not machine-specific.**
  It can run on any system that has a JVM installed. This removes dependency on hardware or operating system.
- **JVM converts bytecode into machine code based on the operating system.**
  The JVM interprets or compiles bytecode into native machine instructions. This happens at runtime.
- **Each OS has its own JVM, but output remains the same.**
  Windows, Linux, and macOS have different JVM implementations. However, the execution result is identical on all platforms.
- **This concept is called Write Once, Run Anywhere (WORA).**
  Developers write the program only once. The same program can run anywhere without recompilation.
- **Platform independence makes Java popular for web and enterprise applications.**
  It reduces development cost and increases portability. This is why Java is widely used in large-scale systems.