THE COMPLETE CORE JAVA COURSE

PRESENTED BY,

KGM TECHNICAL TEAM

SLIDES FOR THEORY LECTURES

KGiSL MicrCollege

Genius as a Service (GaaS)

THE COMPLETE CORE JAVA COURSE

PRESENTED BY,

KGM TECHNICAL TEAM

DAY-01

Welcome Section

LECTURE

Watch before you Start

Introduction to Java

Title: "What is Java?"

Content:

- **Brief History:**

  Developed by Sun Microsystems (1995), now maintained by Oracle.

- **Definition:**

  A high-level, object-oriented, platform-independent programming language.

Key Features:

- **Write Once, Run Anywhere (WORA):** Example: A Java program compiled on Windows can run on Linux without modification.
- Robust, secure, and multithreaded (supports concurrent execution).

Use Cases: Web (Spring Framework), Mobile (Android), Enterprise (Java EE), Embedded Systems (IoT).

Java Development Kit (JDK) Installation

Title: "Setting Up the Development Environment - JDK"

Content:

- **Definition:**
  A development kit containing tools for writing and running Java applications.
- **Components of JDK:**
  - **Compiler (javac):** Converts Java code to bytecode.
  - **JVM:** Executes the bytecode.
  - **Tools:** javadoc, jdb (debugger).

Steps to Install JDK:

1. **Download:** Visit Oracle JDK Downloads.
2. **Install:** Run the installer and follow prompts.
3. **Verify Installation:**
   - java -version
   - javac -version
4. **Set Environment Variable:** Add JAVA_HOME to system variables.

Java Runtime Environment (JRE)

Title: "Understanding JRE"

Content:

- **Definition:**

  JRE allows running Java applications; it includes the JVM and libraries.

- **Key Difference:**

  JDK for development, JRE for execution.

- **Components:**

  - JVM: Executes bytecode.
  - Core Libraries: java.util, java.io, etc.

Why Needed:

- It ensures platform independence by providing a uniform runtime environment.

Integrated Development Environment (IDE) Setup

Title: "Choosing and Setting Up an IDE"

Content:

- **What is an IDE?**

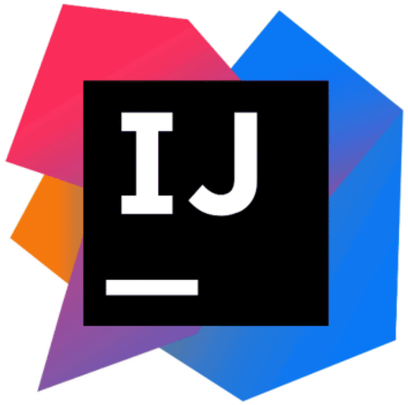
  A tool integrating a code editor, debugger, and compiler for seamless development.

- **Recommended IDEs:**

  - **Eclipse:** Beginner-friendly, supports plugins for various frameworks.
  - **IntelliJ IDEA:** Advanced features, auto-completion, faster debugging.

Integrated Development Environment (IDE)

Java IDEs



Apache
NetBeans IDE

Integrated Development Environment (IDE) Setup

Steps to Set Up IDE:

1. **Download Eclipse:** Eclipse Downloads.

2. Install and Configure JDK: Set the path in IDE preferences.

3. Create a New Project:

- **Eclipse:** File → New → Java Project → Add a new class.

4. Write and Run Example Program:

Benefits: Auto-suggestions, built-in debugger, Maven/Gradle integration.

Compiler in Java

👉 The **compiler** translates Java source code (written in .java files) into bytecode (stored in .class files).

- **Purpose:** Converts human-readable source code into an intermediate, platform-independent format (bytecode).
- **Tool Used:** The javac (Java Compiler) command is used to compile Java programs.
- **Process:**
    - A .java file is written.
    - The javac command compiles it into a .class file containing bytecode.
    - The bytecode is not specific to any operating system or machine, making Java platform-independent.

```
javac MyProgram.java
```

Interpreter in Java

👉 The **interpreter** executes the bytecode line-by-line, making it understandable to the machine.

- **Purpose:** Converts bytecode into machine-specific instructions at runtime.
- **Tool Used:** The Java Virtual Machine (JVM) interprets the bytecode.
- **Process:**
    - The .class file (bytecode) is given to the JVM.
    - The JVM interprets and executes the bytecode line-by-line.
    - If necessary, the Just-In-Time (JIT) compiler may compile portions of the bytecode into native machine code for faster execution.

```
java MyProgram
```

Compiler and Interpreter key Difference

Aspect	Compiler	Interpreter
Role	Converts source code to byte code	Converts byte code to machine-specific code
When it runs	During the compilation phase (before runtime).	During the execution phase (at runtime).
Output	Produces .class files (bytecode).	Executes the program directly.
Tool Used	javac (Java Compiler).	JVM (Java Virtual Machine).

Java Syntax and Structure

Definition:

- Java syntax refers to the set of rules that define the structure of a Java program, including how statements, variables, methods, and classes are written.
- It ensures consistency and readability, allowing programs to be compiled and executed correctly.

Key Elements of Java Syntax:

- **Case Sensitivity:** Java is case-sensitive (Variable and variable are different).
- **Class Declaration:** Every Java program must have at least one class.
- **Main Method:** The entry point of the program (public static void main(String[] args)).
- **Statements:** End with a semicolon (;).
- **Curly Braces {}:** Used to define blocks of code.
- **Comments:**
    - Single-line: // Comment
    - Multi-line: /* Comment */

Example of Java Syntax Structure:

```
// Import necessary libraries (optional)
import java.util.Scanner; // Example of an import

// Class Declaration
public class SyntaxExample {

  // Main Method
  public static void main(String[] args) {
    // Variable Declaration
    int number = 10;
    String message = "Java Syntax Example";

    // Output Statements
    System.out.println("Number: " + number);
    System.out.println("Message: " + message);
  }
}
```

Runnnig a Program after Setup

Procedure to run in command prompt:

- javac HelloWorld.java
- java HelloWorld

Note:

- If you have only JRE installed, you can run.
- But without JDK, you cannot compile.

Hello World Example:

```
// Class Declaration
public class HelloWorld {

  // Main Method
  public static void main(String[] args) {
     // Print "Hello, World!" to the console
     System.out.println("Hello, World!");
  }
}
```

Output:

```
Hello, World!
```

Hello World Program

Definition:

- The "Hello World" program is the simplest Java program, traditionally used as the first program to demonstrate the syntax and structure of Java.
- It prints "Hello, World!" to the console.

Steps to Write a Hello World Program:

1. Create a class (e.g., HelloWorld).
2. Define the main method as the program entry point.
3. Use System.out.println() to print the message to the console.

Explanation of the Code:

- **public class HelloWorld:** Defines a class named HelloWorld.
- **public static void main(String[] args):** The main method, where the program starts execution.
- **System.out.println("Hello, World!");:** Prints "Hello, World!" to the console.

Data Types

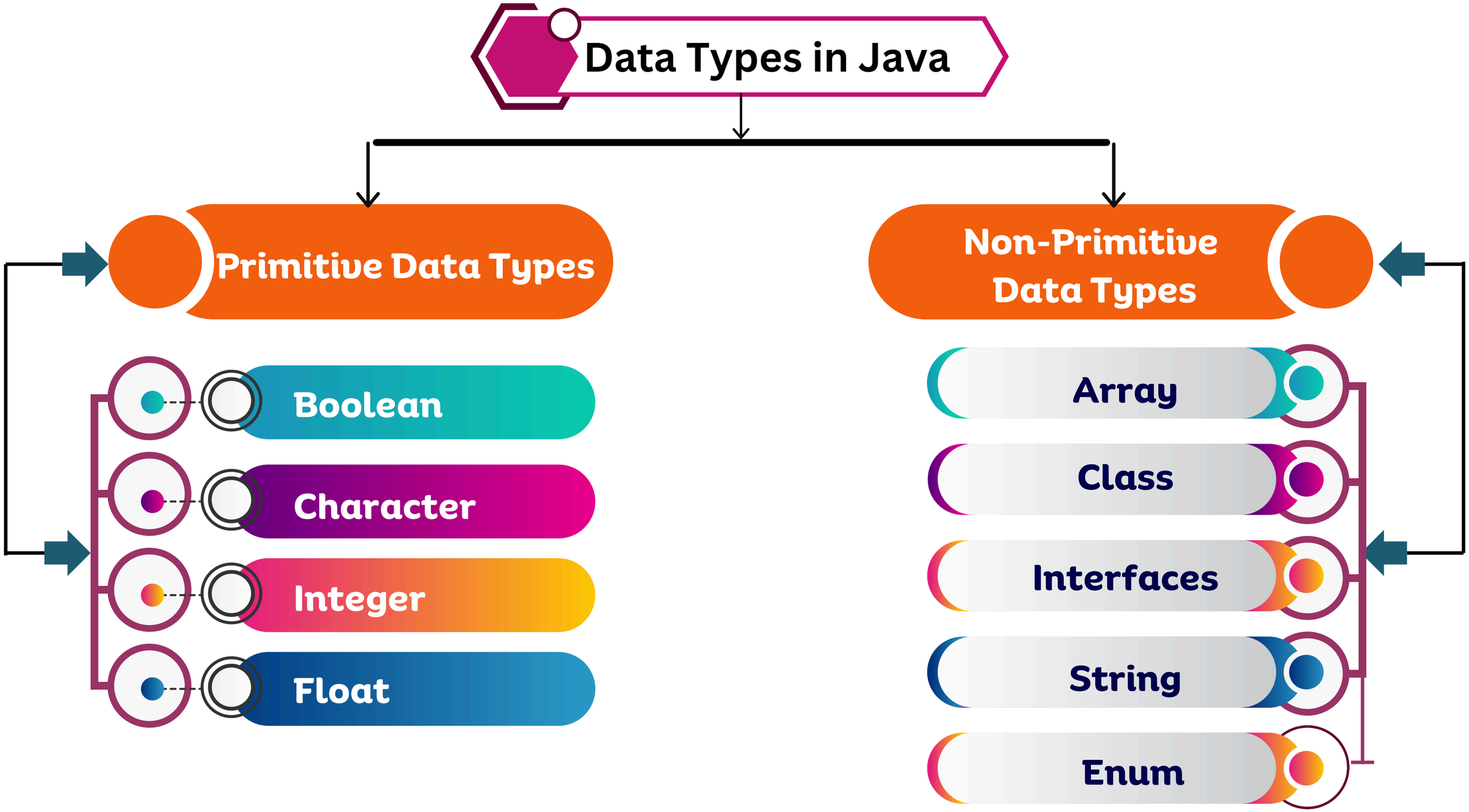
Definition:

- Data types in Java define the type of data a variable can store.
- They determine the memory size and operations that can be performed on the data.

Categories of Data Types:

1. Primitive Data Types.
2. Non-Primitive Data Types.

Data Types


Data Types in Java
Primitive Data Types
Boolean
Character
Integer
Float
Non-Primitive Data Types
Array
Class
Interfaces
String
Enum

Data Types

NAME	DEFAULT	SIZE	TYPE	EXAMPLE
byte	0	8-bit	Integral	byte b = 100;
short	0	16-bit	Integral	short s = 10000;
int	0	32-bit	Integral	int i = 100000;
long	0L	64-bit	Integral	long l = 9999999;
float	0.0f	32-bit	Floating Point	float f = 123.4f;
double	0.0d	64-bit	Floating Point	double d = 12.4;
boolean	false	1-bit	Boolean	boolean b = true;
char	'\u0000'	16-bit	Character	char c = ‘C’;

Data Types

Categories of Data Types:

Primitive Data Types: Predefined in Java and include:

- byte (1 byte), short (2 bytes), int (4 bytes), long (8 bytes) – for integers.
- float (4 bytes), double (8 bytes) – for floating-point numbers.
- char (2 bytes) – for single characters.
- boolean (1 bit) – for true/false values.

Non-Primitive Data Types:

- Strings
- Arrays
- Classes
- Interfaces
- Enums

Example Syntax:

Example Syntax for Primitive Types

```
byte a = 10;
short b = 1000;
int c = 100000;
long d = 10000000000L;

float e = 12.34f;
double f = 123.456;

char g = 'A';
boolean h = true;
```

Examples of Primitive Data Types:

```
public class DataTypesExample {
  public static void main(String[] args) {
    // Primitive Data Types
    int age = 25;       // Integer type
    double salary = 45000.50; // Floating-point type
    char grade = 'A';    // Character type
    boolean isEmployed = true; // Boolean type

    // Output examples
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
    System.out.println("Grade: " + grade);
    System.out.println("Is Employed: " + isEmployed);
  }
}
```

Output for Example :

Output:

```
Age: 25
Salary: 45000.5
Grade: A
Is Employed: true
```

Example Syntax for Non-Primitive Datatypes

String:

- Used to store a sequence of characters.

Example:

```
String text = "Hello, World!";
```

Arrays:

- Used to store multiple values of the same type in a single variable.

Example:

```
int[] numbers = {1, 2, 3, 4, 5};
String[] names = {"Alice", "Bob", "Charlie"};
```

Example Syntax for Non-Primitive Datatypes

Classes:

- Used to define custom data types that contain attributes and methods.

Example:

```
public class Car {
  String color;
  int year;

  public Car(String color, int year) {
    this.color = color;
    this.year = year;
  }
}
Car myCar = new Car("Red", 2021);
```

Example Syntax for Non-Primitive Datatypes

Interface:

- Defines a contract that classes can implement.

Example:

```
interface Animal {
  void makeSound();
}
```

Enum:

- Used to define a fixed set of constants.

Example:

```
enum Days {
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY
}
Days today = Days.MONDAY;
```

Variables

Definition:

- A variable is a container that stores data in memory.
- It has a data type, name, and value.
- Variables can change their value during program execution.

Syntax to Declare a Variable:

```
dataType variableName = value;
```

Rules for Naming Variables:

- Must start with a letter or _ (underscore).
- Cannot use Java keywords (e.g., class, public).
- Case-sensitive (myVar and myvar are different).

Examples of Variables:

```
public class VariablesExample {
  public static void main(String[] args) {
    // Declare variables
    int number = 10;      // Integer variable
    String name = "Alice"; // String variable

    // Change the value of a variable
    number = 20;

    // Output variables
    System.out.println("Number: " + number);
    System.out.println("Name: " + name);
  }
}
```

Output for Example :

Output:

```
Number: 20
Name: Alice
```

Variables Types

Types:

1. Local Variable
2. Instance Variable
3. Static Variable

Local Variables:

- Declared within a method or a block and accessible only within that scope.

Example:

```
public void myMethod() {
  int localVar = 10; // Local variable
}
```

Variables Types

Instance Variables:

- Defined within a class but outside any method, accessible by all methods in the class (each object has its own copy).

Example:

```
public class Car {
  String color; // Instance variable
}
```

Variables Types

Static Variables:

- Declared with the static keyword, shared among all instances of a class.

Example:

```
public class Car {
  static int numberOfWheels = 4; // Static variable
}
```

Examples of Variable Types:

```
public class VariableTypes {
  int d = 30; //Instance variable
  static int a = 10; //Static Variable
      static void display(){
         int b = 25; //Local Variable
        //return b;
      }
  public static void main(String[] args){
     int c = 55; // Local Variable
     System.out.println(a);
     System.out.println(c);
     display();
  }
}
```

Constants

Definition:

- A constant is a variable whose value cannot be changed once it is assigned.
- Constants in Java are declared using the final keyword.

Syntax to Declare a Variable:

final dataType CONSTANT_NAME = value;

Why Use Constants?

- Improve code readability and maintainability.
- Prevent accidental changes to critical values.

Examples of Constants:

```
public class ConstantsExample {
  public static void main(String[] args) {
    // Declare a constant
    final double PI = 3.14159; // Constant for the value of pi

    // Try to change the constant (uncommenting the next line will cause an error)
    // PI = 3.14;

    // Output the constant
    System.out.println("Value of PI: " + PI);
  }
}
```

Output for Example :

Output:

Value of PI: 3.14159

Operators

Operators in Java

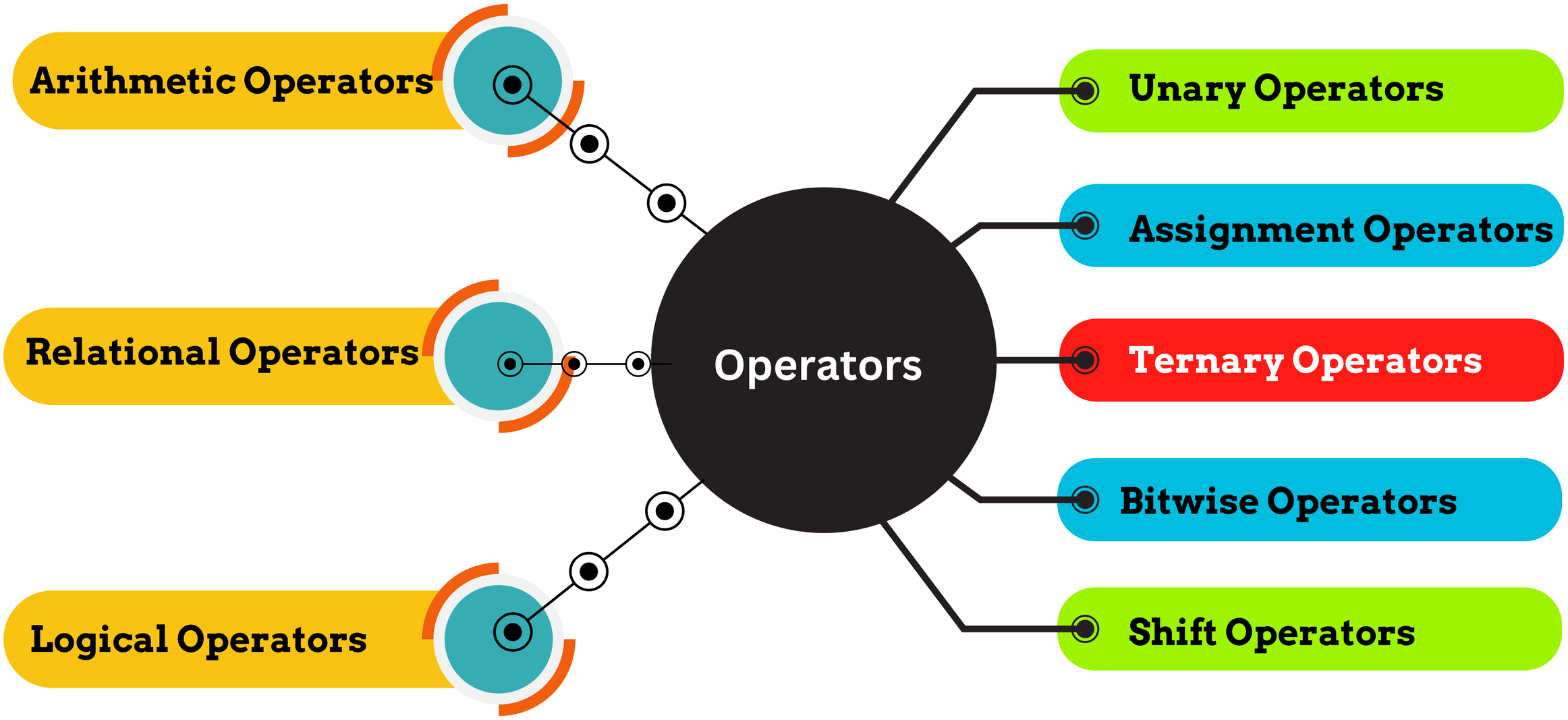
Definition:

- Operators in Java are special symbols or keywords used to perform operations on variables and values.
- Operators are classified into various types based on their functionality.

Types Covered:

1. **Arithmetic Operators:** Perform basic mathematical operations.
    - + (Addition)
    - - (Subtraction)
    - * (Multiplication)
    - / (Division)
    - % (Modulus)

Operators in Java



Examples of Arithmetic Operators:

```
public class ArithmeticOperators {
  public static void main(String[] args) {
    int a = 10, b = 5;

    System.out.println("Addition: " + (a + b));
    System.out.println("Subtraction: " + (a - b));
    System.out.println("Multiplication: " + (a * b));
    System.out.println("Division: " + (a / b));
    System.out.println("Modulus: " + (a % b));
  }
}
```

Output for Example :

Output:

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2
Modulus: 0
```

Relational Operators

2. **Relational Operators:** Compare two values and return a boolean result.
    - == (Equal to)
    - != (Not equal to)
    - < (Less than)
    - > (Greater than)
    - <= (Less than or equal to)
    - >= (Greater than or equal to)

Examples of Relational Operators:

```
public class RelationalOperators {
  public static void main(String[] args) {
    int a = 10, b = 5;

    System.out.println("a == b: " + (a == b));
    System.out.println("a != b: " + (a != b));
    System.out.println("a > b: " + (a > b));
    System.out.println("a < b: " + (a < b));
    System.out.println("a >= b: " + (a >= b));
    System.out.println("a <= b: " + (a <= b));
  }
}
```

Output for Example :

Output:

```
a == b: false
a != b: true
a > b: true
a < b: false
a >= b: true
a <= b: false
```

Logical Operators

3. **Logical Operators:** Combine multiple boolean expressions or values.
    - && (Logical AND)
    - || (Logical OR)
    - ! (Logical NOT)

Example:

```
public class LogicalOperators {
  public static void main(String[] args) {
     boolean condition1 = true, condition2 = false;

     System.out.println("condition1 && condition2: " + (condition1 && condition2));
     System.out.println("condition1 || condition2: " + (condition1 || condition2));
     System.out.println("!condition1: " + (!condition1));
  }
}
```

Output for Example :

Output:

condition1 && condition2: false
condition1 || condition2: true
!condition1: false

Unary Operators

4. **Unary Operators:** Unary operators operate on a single operand.
    - + (Positive)
    - - (Negative)
    - ++ (Increment)
    - -- (Decrement)
    - ! (Logical NOT)

Examples of Unary Operators:

```
public class UnaryOperators {
    public static void main(String[] args) {
        int a = 10;
        boolean b = true;

        System.out.println("+a: " + (+a));
        System.out.println("-a: " + (-a));
        System.out.println("++a: " + (++a)); // Pre-increment
        System.out.println("a++: " + (a++)); // Post-increment
        System.out.println("Value of a after a++: " + a);
        System.out.println("--a: " + (--a)); // Pre-decrement
        System.out.println("a--: " + (a--)); // Post-decrement
        System.out.println("Value of a after a--: " + a);
        System.out.println("!b: " + (!b));
    }
}
```

Output for Example :

Output:

```
+a: 10
-a: -10
++a: 11
a++: 11
Value of a after a++: 12
--a: 11
a--: 11
Value of a after a--: 10
!b: false
```

Assignment Operators

5. **Assignment Operators:** Assignment operators assign values to variables.
    - =
    - +=
    - -=
    - *=
    - /=
    - %=

Examples of Assignment Operators:

```
public class AssignmentOperators {
    public static void main(String[] args) {
        int a = 10;

        a += 5; // a = a + 5
        System.out.println("a += 5: " + a);
        a -= 3; // a = a - 3
        System.out.println("a -= 3: " + a);
        a *= 2; // a = a * 2
        System.out.println("a *= 2: " + a);
        a /= 4; // a = a / 4
        System.out.println("a /= 4: " + a);
        a %= 3; // a = a % 3
        System.out.println("a %= 3: " + a);
    }
}
```

Output for Example :

Output:

```
a += 5: 15
a -= 3: 12
a *= 2: 24
a /= 4: 6
a %= 3: 0
```

Ternary Operators

6. **Ternary Operators:** The ternary operator is a shorthand for if-else.
    - condition ? value_if_true : value_if_false.

Example:

```
public class TernaryOperator {
   public static void main(String[] args) {
      int a = 10, b = 5;

      String result = (a > b) ? "a is greater" : "b is greater";
      System.out.println("Result: " + result);
   }
}
```

Output:

```
Result: a is greater
```

Bitwise Operators

7. **Bitwise Operators:** Bitwise operators work on binary representations of numbers.
   - & (AND)
   - | (OR)
   - ^ (XOR)
   - ~ (NOT)

Bitwise AND (&):

A	B	A&B
1	1	1
1	0	0
0	1	0
0	0	0

Logical AND (&&):

A	B	A&&B
T	T	T
T	F	F
F	T	F
F	F	F

System.out.println(38 & 23) == ???

	32	16	8	4	2	1
38_{10}	1	0	0	1	1	0
23_{10}	0	1	0	1	1	1
???	0	0	0	1	1	0

Examples of Bitwise Operators:

```
public class BitwiseOperators {
    public static void main(String[] args) {
        int a = 5, b = 3; // Binary: a = 0101, b = 0011

        System.out.println("a & b: " + (a & b)); // AND
        System.out.println("a | b: " + (a | b)); // OR
        System.out.println("a ^ b: " + (a ^ b)); // XOR
        System.out.println("~a: " + (~a));      // NOT
    }
}
```

Output for Example :

Output:

```
a & b: 1
a | b: 7
a ^ b: 6
~a: -6
```

Shift Operators

8. **Shift Operators:** Shift operators shift bits to the left or right.
    - << (Left shift)
    - >> (Right shift)
    - >>> (Unsigned right shift)

Example:

```
public class ShiftOperators {
    public static void main(String[] args) {
        int a = 8; // Binary: 00001000

        System.out.println("a << 2: " + (a << 2)); // Left shift
        System.out.println("a >> 2: " + (a >> 2)); // Right shift
        System.out.println("a >>> 2: " + (a >>> 2)); // Unsigned right shift
    }
}
```

Output for Example :

Output:

```
a << 2: 32
a >> 2: 2
a >>> 2: 2
```

Input and Output in Java

Definition:

- **Input:** Reading data provided by the user during program execution.
- **Output:** Displaying results or messages to the user.

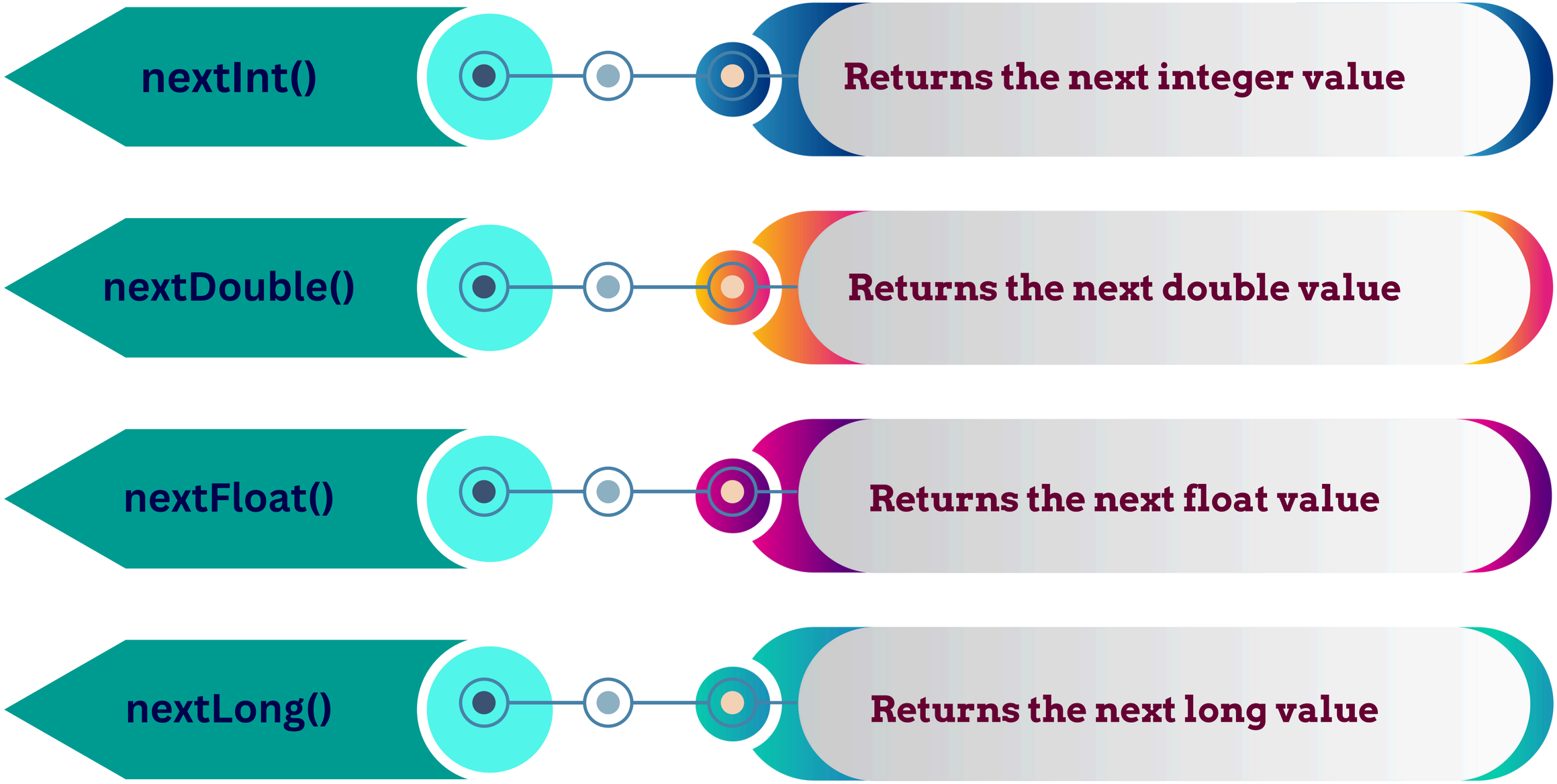
Java Classes for Input/Output:

1. **System.out.println()** – Prints messages to the console (output).
2. **Scanner Class** – Reads input from the user.

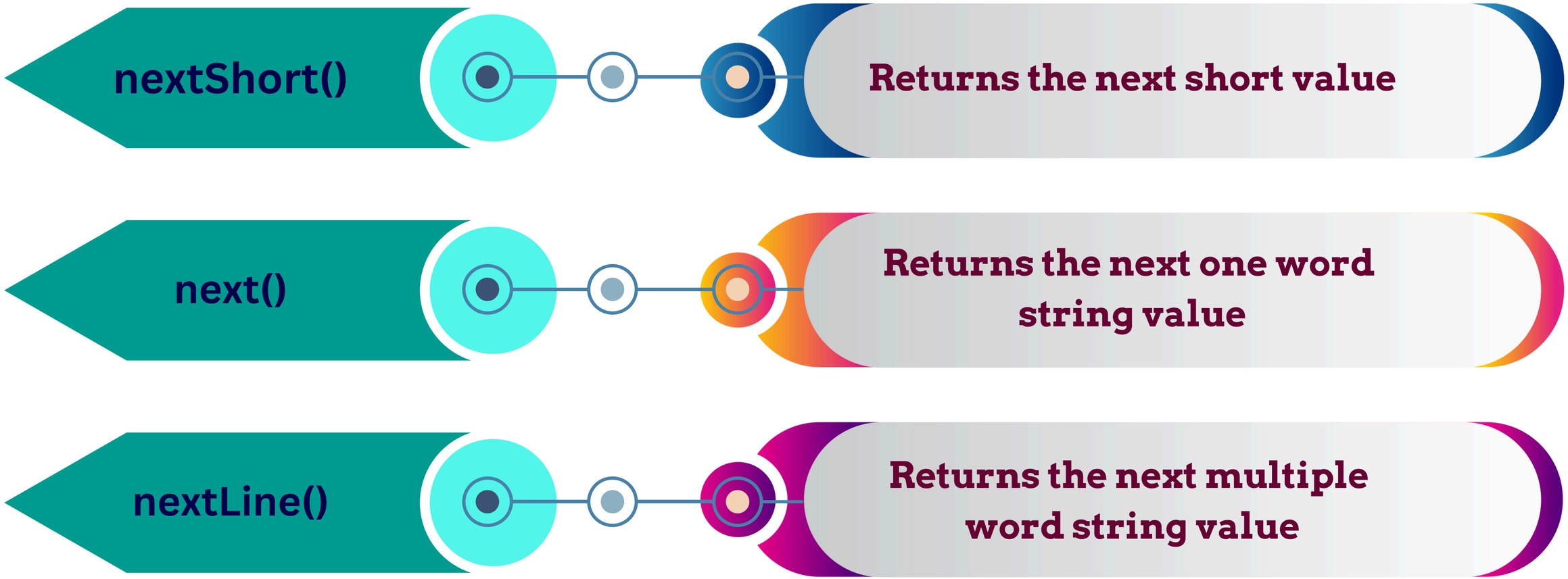
Syntax for Scanner Class:

```
Scanner scanner = new Scanner(System.in);
dataType variableName = scanner.nextDataType();
```

Input and Output in Java



Input and Output in Java



Examples of I/O:

```
import java.util.Scanner;

public class InputOutputExample {
  public static void main(String[] args) {
    // Create Scanner object
    Scanner scanner = new Scanner(System.in);
    // Input
    System.out.println("Enter your name: ");
    String name = scanner.nextLine(); // Reads a string
    System.out.println("Enter your age: ");
    int age = scanner.nextInt(); // Reads an integer
   // Output
    System.out.println("Hello, " + name + "! You are " + age + " years old.");
  }
}
```

Input and Output in Java

Steps to Execute:

1. Compile the program: javac InputOutputExample.java.
2. Run the program: java InputOutputExample.
3. Provide input when prompted.

Output:

```
Enter your name: Alice
Enter your age: 25
Hello, Alice! You are 25 years old.
```

KGiSL MicroCollege

Genius as a Service (GaaS)

THE COMPLETE CORE JAVA COURSE

PRESENTED BY,

KGM TECHNICAL TEAM

DAY-01 COMPLETED 🎉