import java.util.Scanner;

public class NamePrinter {

// Field (attribute)

String name;

// Method to input the name and print it

public void getNameAndPrint() {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter their name

System.out.print("Enter your name: ");

name = scanner.nextLine(); // Read the user input

// Print the name

System.out.println("Hello, " + name + "!");

// Close the scanner

scanner.close();

}

public static void main(String[] args) {

// Create an object of the NamePrinter class

NamePrinter user = new NamePrinter();

// Call the method to input the name and print it

user.getNameAndPrint();

}

}

import java.util.Scanner;

public class NamePrinter {

// Field (attribute)

String name;

// Method to print the name

public void printName() {

System.out.println("Hello, " + name + "!");

}

public static void main(String[] args) {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter their name

System.out.print("Enter your name: ");

String userName = scanner.nextLine(); // Read the user input

// Create an object of NamePrinter and pass the user name

NamePrinter user = new NamePrinter();

user.name = userName; // Set the name field from the main method

// Call the method to print the name

user.printName();

// Close the scanner

scanner.close();

}

}

public class Calculator {

// Method with parameters 'a' and 'b'

public int add(int a, int b) { // 'a' and 'b' are parameters

return a + b;

}

public static void main(String[] args) {

Calculator calc = new Calculator();

// Passing 5 and 3 as arguments

int result = calc.add(5, 3); // 5 and 3 are arguments

System.out.println("The result is: " + result);

}

}

public class Calculator {

// Fields (attributes)

int a;

int b;

// Constructor to initialize the values of 'a' and 'b'

public Calculator(int a, int b) {

this.a = a; // Initialize 'a' with the passed argument

this.b = b; // Initialize 'b' with the passed argument

}

// Method to add 'a' and 'b'

public int add() {

return a + b; // Use the values stored in 'a' and 'b'

}

public static void main(String[] args) {

// Create a Calculator object and pass values to the constructor

Calculator calc = new Calculator(5, 3); // 5 and 3 are passed as arguments to the constructor

// Call the add method to get the result

int result = calc.add(); // No arguments needed, uses the values from the constructor

// Print the result

System.out.println("The result is: " + result);

}

}

**// this is for accessing data with out arrlength**

For accessing array int arr[]=(10,20,30,40,50)

For(data:arr)

Print data

**// for break and continue**

**Break-** skip the one parent loop

**Continue just**- skip the down process

**Return-** return from where it comes from

**// calling function**

Class Car {

Static void carName(){  
System.out.println(“Ferrari”);

}

Public void maxspeed(){

System.out.println(“360”);

}

}

Public class FirstJava{

Public static void main(Strig)

Car.carname // accessing with out object because this function is static

Car c= new Car(); // two car should be same

c.maxspeed // accessing as object because this function in public

}

**// object dynamically allocate memory in run time**

i.e Box boxVar; //it is null

boxVar = new Box(); // now it has memory

// p**rivate**

**Inner classes**

**Public ,private ,protected ,inheritance,extended ,static**

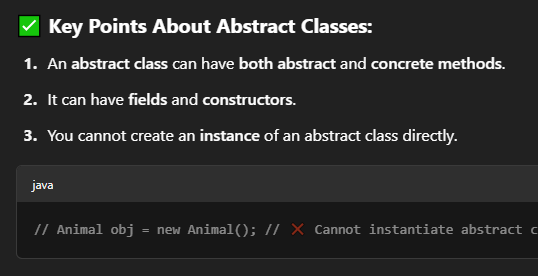
**Construcor can be different name ??**

**If we use extend keyword that is only sub class..**

**In extend we cannot use private of class**

**Child only access parent but parent cannot access child in heritance.**

Abstraction---------------------------------------------------------



class Animal {

String name = "Animal";

}

class Dog extends Animal {

String name = "Dog";

void display() {

System.out.println(super.name); // prints "Animal"

System.out.println(this.name); // prints "Dog"

}

}

Static---------------------------------------------------------------------------------------

class Counter {

// Static variable

private static int count = 0;

// Static method to increment count

public static void increment() {

count++;

}

// Static method to get the current count

public static int getCount() {

return count;

}

}

public class Main {

public static void main(String[] args) {

// Calling static methods without creating an instance of Counter

Counter.increment(); // count becomes 1

Counter.increment(); // count becomes 2

System.out.println("Count: " + Counter.getCount()); // Outputs: Count: 2

}

}

\*\*\*\* without making an object of class u can call a static

--------------------final word

class Circle {

// Final variable

public static final double PI = 3.14159;

public double calculateArea(double radius) {

return PI \* radius \* radius; // PI cannot be changed

}

}

public class Main {

public static void main(String[] args) {

Circle circle = new Circle();

System.out.println("Area: " + circle.calculateArea(5)); // Outputs area using final PI

}

}

\*\*\*\*\* as a constant keyword-------------------------------------------

\*\*\*\*\*\*The FinalClass cannot be extended because it is declared as final. If we try to create a subclass of FinalClass, it will cause a **compile-time error**.

Access modifier

-------------------------------------privae -----------------------------------------------

class BankAccount {

private double balance;

public void deposit(double amount) {

if (amount > 0) {

balance += amount; // Accessed within the same class

}

}

public double getBalance() {

return balance; // Accessed via a public method

}

}

public class Main {

public static void main(String[] args) {

BankAccount account = new BankAccount();

// account.balance = 1000; // Error: balance is private

account.deposit(500); // Access allowed through public method

System.out.println("Balance: " + account.getBalance()); // Access allowed through public method

}

}

A black screen with white text

AI-generated content may be incorrect.

**✅ Quick Check (Mini Quiz)**

1. What is the result of 10 % 3?
2. What does == do in Java?
3. How do you write a condition that checks if a number is **not equal to 100**?
4. Write a ternary operator to check if a person is **eligible to vote** (age ≥ 18).
5. Will this print true?

java

CopyEdit

String name = "Java";

System.out.println(name instanceof Object);

Java topics

**1. Introduction to Java**

* History and Features of Java
* Java Editions (SE, EE, ME)
* JVM, JRE, and JDK
* Java Program Structure
* Compilation and Execution

**2. Data Types and Variables**

* Primitive Data Types
* Reference Data Types
* Type Conversion and Casting
* Constants and Literals
* Variable Scope and Lifetime

| * **Data Type** | * **Size** | * **Example** |
| --- | --- | --- |
| * byte | * 1 byte | * byte b = 100; |
| * short | * 2 bytes | * short s = 1000; |
| * int | * 4 bytes | * int a = 5000; |
| * long | * 8 bytes | * long l = 123456789L; |
| * float | * 4 bytes | * float f = 3.14f; |
| * double | * 8 bytes | * double d = 99.99; |
| * char | * 2 bytes | * char c = 'A'; |
| * boolean | * 1 bit | * boolean flag = true; |

**3. Operators and Expressions**

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment and Unary Operators
* Ternary Operator
* Operator Precedence

**4. Control Flow Statements**

* Decision Making: if, if-else, switch
* Loops: for, while, do-while
* Jump Statements: break, continue, return

**5. Object-Oriented Programming (OOP)**

* Class and Object
* Constructors
* this Keyword
* Inheritance
* Method Overloading and Overriding
* super Keyword
* Polymorphism
* Abstraction (abstract, interfaces)
* Encapsulation
* Static Members
* Final Keyword

**6. Arrays and Strings**

* One-Dimensional and Multi-Dimensional Arrays
* Array Operations
* String Class and Methods
* StringBuilder and StringBuffer

**7. Packages and Access Modifiers**

* Built-in Packages (java.util, java.io, etc.)
* User-defined Packages
* import Statement
* Access Modifiers: public, private, protected, default

**8. Exception Handling**

* Types of Exceptions
* try, catch, finally
* throw and throws
* Custom Exceptions

**9. Java Collections Framework (JCF)**

* List (ArrayList, LinkedList)
* Set (HashSet, LinkedHashSet, TreeSet)
* Map (HashMap, TreeMap, LinkedHashMap)
* Queue and Stack
* Iterators and Enhanced for-loop

**10. File Handling**

* File Class
* Reading and Writing Files
* FileReader, FileWriter
* BufferedReader, BufferedWriter
* Serialization and Deserialization

**11. Multithreading and Concurrency**

* Thread Class and Runnable Interface
* Thread Lifecycle
* Synchronization
* Inter-thread Communication
* Executor Framework

**12. GUI Programming (AWT and Swing)**

* AWT Components
* Swing Components (JFrame, JPanel, JButton, etc.)
* Event Handling
* Layout Managers

**13. JavaFX (Modern GUI Toolkit)**

* Application Lifecycle
* Scene Graph and UI Controls
* FXML and CSS
* Event Handling in JavaFX

**14. Networking in Java**

* InetAddress, Socket, ServerSocket
* URL and HttpURLConnection
* Client-Server Communication

**15. JDBC (Java Database Connectivity)**

* Connecting to a Database
* Statements and ResultSet
* PreparedStatement
* CRUD Operations
* Connection Pooling

**16. Annotations and Reflection**

* Built-in Annotations (@Override, @Deprecated, etc.)
* Custom Annotations
* Reflection API

**17. Lambda Expressions and Streams (Java 8+)**

* Functional Interfaces
* Lambda Expressions
* Stream API (map, filter, reduce)
* Method References

**18. Modules (Java 9+)**

* Introduction to Modules
* module-info.java
* Module Dependency

**19. Java Development Tools**

* IDEs: IntelliJ IDEA, Eclipse, NetBeans
* Build Tools: Maven, Gradle
* Debugging and Testing (JUnit)

**20. Best Practices and Design Patterns**

* SOLID Principles
* Common Design Patterns (Singleton, Factory, Observer, etc.)
* Clean Code and Documentation