Q1: What is a class? What is an object?

Ans: A class is a blueprint/template that defines properties (variables) and behaviors (methods). An object is an instance of a class that uses these properties and behaviors.

Q2: Explain the concept of OOPS?

Ans: OOPS (Object-Oriented Programming System) is a paradigm that organizes code into objects. It is based on principles like Encapsulation, Inheritance, Polymorphism, and Abstraction.

Q3: What are the advantages of OOPS over procedural programming?

Ans: Reusability, modularity, easier debugging, scalability, data security, and real-world modeling.

Q4: What is Java?

Ans: Java is a high-level, object-oriented, platform-independent programming language that follows the principle "Write Once, Run Anywhere."

Q5: What is the extension of a java file? Explain the features of Java.

Ans: Extension is .java.

Features: Platform-independent, Object-oriented, Secure, Robust, Multithreaded, Distributed, High performance.

Q6: Write a program to print a pyramid of star patterns.

Ans:

```
class StarPyramid {

 public static void main(String[] args) {

 int n = 5;

 for(int i=1; i<=n; i++) {

 for(int j=i; j<n; j++) System.out.print(" ");

 for(int k=1; k<=2*i-1; k++) System.out.print("*");

 System.out.println();

 }

 }

}
```

Q7: Explain the features of OOPS.. Discuss the five pillars of OOPS.

Ans: Features → Modularity, Reusability, Security, Abstraction.

Five pillars → Encapsulation, Abstraction, Inheritance, Polymorphism, Class & Object.

Q8: Write the comparison between Abstraction and Encapsulation.

Ans:

Abstraction → Hides implementation details, shows only functionality. (What to do)

Encapsulation → Wrapping of data + methods into a single unit (class). (How to protect data)

Q9: Write the name of a pure object-oriented programming language. Compare between Structured programming and OOP.

Ans: Smalltalk is a pure OOP language.

Structured: Top-down, functions, less secure.

OOP: Bottom-up, objects, encapsulation & reusability.

Q10: Write a program in Java to find the Longest and second longest compound word from a text file.

Ans: (Skeleton code, short)

```
import java.io.*;

import java.util.*;

class LongestCompound {

 public static void main(String[] args) throws Exception {

 List<String> words = new ArrayList<>();

 Scanner sc = new Scanner(new File("words.txt"));

 while(sc.hasNext()) words.add(sc.next());

 words.sort((a,b)->b.length()-a.length());

 System.out.println("Longest: " + words.get(0));

 System.out.println("Second Longest: " + words.get(1));

 }

}
```

Q11: What is a top down approach? What is the bottom up approach?

Ans:

Top-down → Start from main system → break into sub-modules.

Bottom-up → Start from smaller modules → integrate to form system.

Q12: What is the bottom up approach?

Ans: Build system from small components (objects) → combine into larger units. Java follows bottom-up (OOP).

Q13: What are the main differences between the Java platform and other platforms?

Ans: Java compiles to bytecode which runs on JVM (platform-independent), unlike C/C++ which compiles to machine code (platform-dependent).

Q14: What is byte code? What is JVM?

Ans:

Bytecode → Intermediate code generated after Java compilation.

JVM (Java Virtual Machine) → Executes bytecode, makes Java platform-independent.

Q15: Write a program to count the total number of vowels and consonants in a string.

Ans:

```
class VowelConsonant {

 public static void main(String[] args) {

 String s = "Hello World";

 s = s.toLowerCase();

 int v=0,c=0;

 for(char ch: s.toCharArray()) {

 if("aeiou".indexOf(ch)>=0) v++;

 else if(ch>='a' && ch<='z') c++;

 }

 System.out.println("Vowels: " + v + ", Consonants: " + c);

 }

}
```

Q16: **What is a JIT compiler? State the steps to compile and run a Java program.**
Ans: **JIT (Just-In-Time) compiler** converts bytecode into machine code at runtime for faster execution.
Steps:

1. Write program → MyClass.java
2. Compile → javac MyClass.java (creates .class)
3. Run → java MyClass

Q17: **Program to count the total number of vowels and consonants in a string.**
Ans:

```
class CountVC {

   public static void main(String[] args) {

      String s = "ICRA Analytics";

      s = s.toLowerCase();

      int v=0,c=0;

      for(char ch: s.toCharArray()){

         if("aeiou".indexOf(ch)>=0) v++;

         else if(ch>='a' && ch<='z') c++;

      }

      System.out.println("Vowels="+v+", Consonants="+c);

   }

}
```

Q18: **What gives Java its 'write once and run anywhere' nature? Write a program to calculate the sum of diagonal of a matrix.**
Ans: Bytecode + JVM make Java platform-independent.

```
class DiagonalSum {

   public static void main(String[] args) {

      int[][] a = {{1,2,3},{4,5,6},{7,8,9}};

      int sum=0;

      for(int i=0;i<a.length;i++) sum += a[i][i];

      System.out.println("Sum = "+sum);

   }

}
```

Q19: **State the differences between JDK, JRE, JVM.**
Ans:

- **JDK**: Development kit (compiler, debugger, JRE).
- **JRE**: Runtime environment (JVM + libraries).
- **JVM**: Virtual machine that runs bytecode.

Q20: **Why is Java called platform independent?**

Ans: Because compiled bytecode runs on JVM, which is available on all platforms.

Q21: **Briefly explain different types of data types in Java.**

Ans:

- **Primitive**: byte, short, int, long, float, double, char, boolean.
- **Non-primitive**: String, Arrays, Classes, Objects.

Q22: **What is the difference between a class and an object?**

Ans: Class = blueprint; Object = instance of a class with real values.

Q23: **If I don't provide any arguments on the command line, then what will the value stored in the String array passed into the main() method, empty or NULL?**

Ans: It will be an **empty array**, not null.

Q24: **What is inheritance?**

Ans: Mechanism where one class acquires properties & behaviors of another class using extends.

Q25: **What is encapsulation?**

Ans: Wrapping of data (variables) and methods in a single unit (class), restricting direct access using private fields + getters/setters.

Q26: **What is polymorphism?**

Ans: Ability of one entity to take many forms.

- Compile-time → Method overloading.
- Runtime → Method overriding.

Q27: **What is abstraction?**

Ans: Hiding implementation details and showing only essential features. Achieved using **abstract classes** and **interfaces**.

Q28: **What are access modifiers? Explain the usage of them in JAVA.**

Ans: Keywords that set access level of classes/members:

- public: accessible everywhere.
- protected: accessible within package & subclasses.
- default: accessible within package.
- private: accessible only in class.

Q29: **What are the different types of variables in Java?**

Ans:

- **Local** → declared inside methods.
- **Instance** → per object, non-static.
- **Static** → shared by all objects of class.

Q30: **What are control structures in Java?**

Ans: Structures that control program flow:

- Decision-making: if, switch.
- Looping: for, while, do-while, enhanced for.
- Jumping: break, continue, return.

Q31: **What is an enhanced for loop in Java?**

Ans: A simplified for-each loop used to iterate over arrays/collections.

```
for(int x : arr) System.out.println(x);
```

Q32: **What is a reference variable in Java?**

Ans: A variable that holds the address (reference) of an object instead of the object itself.

Q33: **What is function overloading?**

Ans: Defining multiple methods with the same name but different parameter lists (compile-time polymorphism).

Q34: **What is function overriding?**

Ans: Redefining a method of the parent class in the subclass with the same signature (runtime polymorphism).

Q35: **How function overloading differs from function overriding?**

Ans:

- **Overloading** → same class, different parameters, compile-time.
- **Overriding** → subclass, same parameters, runtime.

Q36: **Define Constructor.**

Ans: A special method in a class with the same name as the class, used to initialize objects.

Q37: **How many types of Constructors are in Java?**

Ans: Two types → **Default constructor** and **Parameterized constructor**.

Q38: **Write a Java Program to Copy the values from one object to another Object. Program to generate all factors of a number.**

Ans:

```
class Student {

  int id; String name;

  Student(int i, String n){ id=i; name=n; }

  Student(Student s){ id=s.id; name=s.name; } // copy constructor

}
```

Factors program:

```
class Factors {

  public static void main(String[] args){

    int n=12;

    for(int i=1;i<=n;i++)

      if(n%i==0) System.out.print(i+" ");

  }
}
```

Q39: **Is there any method to call a subclass constructor from a superclass constructor?**

Ans: No. Superclass constructor cannot directly call subclass constructor. Flow is always parent → child.

Q40: **Can we have a constructor in the Interface? Justify.**

Ans: No, because interfaces cannot be instantiated; only implemented by classes.

Q41: **Explain Constructor Chaining (give example).**

Ans: Calling one constructor from another in the same class (using this()) or superclass (using super()).

```
class A {

   A(){ this(10); }

   A(int x){ System.out.println(x); }

}
```

Q42: **What is a private constructor?**

Ans: A constructor declared private to restrict object creation (e.g., Singleton pattern).

Q43: **Why constructors in Java cannot be static?**

Ans: Because constructors are invoked to create objects, while static belongs to class, not instance.

Q44: **Can we make a constructor final?**

Ans: No. final prevents overriding, but constructors are not inherited/overridden.

Q45: **Can we make a constructor abstract?**

Ans: No. Abstract means "incomplete, must be overridden," but constructors are never inherited/overridden.

Q46: **What is No-arg constructor?**

Ans: A constructor without parameters, initializes objects with default values.

Q47: **When do we need Constructor Overloading?**

Ans: When we want multiple ways to initialize an object with different sets of data.

Q48: **Do we have destructors in Java?**

Ans: No. Java has **garbage collector** which automatically destroys unused objects.

Q49: **What will happen when a constructor is declared as protected?**

Ans: It can only be accessed within the same package and by subclasses.

Q50: **Why is the constructor name similar to the class name?**

Ans: To tell the JVM which class's object is being initialized.

Q51: **Why is the return type not allowed for the constructor?**

Ans: Because constructors don't return values, they initialize objects implicitly.

Q52: **What is an array in Java? Write a program to sum even numbers from an array.**

Ans: Array = collection of similar data elements stored in contiguous memory.

```
class SumEven {

    public static void main(String[] args){

        int[] arr={1,2,3,4,5,6};

        int sum=0;

        for(int x:arr) if(x%2==0) sum+=x;

        System.out.println("Sum="+sum);

    }

}
```

Q53: **What are the types of an array? Write a program to generate Fibonacci Series in Java.**
Ans: Types → **Single-dimensional** and **Multi-dimensional** arrays.

```
class Fibonacci {

    public static void main(String[] args){

        int n=7, a=0,b=1,c;

        for(int i=0;i<n;i++){

            System.out.print(a+" ");

            c=a+b; a=b; b=c;

        }

    }

}
```

Q54: **Is it possible to declare array size as negative? Write Merge Sort in Java.**
Ans: No, negative size gives **NegativeArraySizeException**.
Merge Sort (short version):

```
class MergeSort {

    void merge(int arr[],int l,int m,int r){

        int n1=m-l+1,n2=r-m;

        int L[]=new int[n1],R[]=new int[n2];

        for(int i=0;i<n1;i++) L[i]=arr[l+i];

        for(int j=0;j<n2;j++) R[j]=arr[m+1+j];
        int i=0,j=0,k=l;

        while(i<n1&&j<n2) arr[k++]= (L[i]<=R[j])?L[i++]:R[j++];

        while(i<n1) arr[k++]=L[i++];

        while(j<n2) arr[k++]=R[j++];

    }

    void sort(int arr[],int l,int r){

        if(l<r){ int m=(l+r)/2; sort(arr,l,m); sort(arr,m+1,r); merge(arr,l,m,r); }

    }

}
```

Q55: **What is the difference between int array[] and int[] array? Write program to print Prime numbers between 120–180.**
Ans: Both are same, just different syntax.

```
class Prime {

    public static void main(String[] args){

        for(int n=120;n<=180;n++){

            int f=1;

            for(int i=2;i<=Math.sqrt(n);i++) if(n%i==0){f=0;break;}

            if(f==1) System.out.print(n+" ");

        }

    }

}
```

Q56: **How to copy an array in Java? Find unique elements in an array and arrange them.**
Ans: Copy → System.arraycopy(src,0,dest,0,length);
Unique elements → Use HashSet to remove duplicates.

Q57: **What is the default value of the array? Can you store various data type in a single array in Java?**
Ans: Default = 0 for numbers, false for boolean, null for objects. Cannot store multiple datatypes in one array; use Object[].

Q58: **What do you understand by the jagged array? What happens if we declare an array without assigning the size? Write method for grouping string by n size.**
Ans: **Jagged array** → array of arrays with different lengths. Without size = compilation error.
Example method:

```
static String[] splitByN(String s,int n){
    int len=s.length();

    int parts=(len+n-1)/n;

    String[] res=new String[parts];

    for(int i=0;i<parts;i++){

        int start=i*n, end=Math.min(start+n,len);

        res[i]=s.substring(start,end);

    }

    return res;

}
```

Q59: **Write code to implement stack using array.**
Ans:

```
class Stack {

    int top=-1, arr[]=new int[10];

    void push(int x){ if(top<9) arr[++top]=x; }

    int pop(){ return (top>=0)?arr[top--]:-1; }

}
```

Q60: **Can we declare array size as negative? When ArrayIndexOutOfBoundsException occurs? Write program for age check.**
Ans: No → Negative size gives **NegativeArraySizeException**.
ArrayIndexOutOfBoundsException → occurs when accessing invalid index.

```
import java.util.*;

class AgeCheck {

    public static void main(String[] args){

        Scanner sc=new Scanner(System.in);

        int age=sc.nextInt();

        if(age>18) System.out.println("yes");

        else if(age>=0) System.out.println("no");

        else System.out.println("Invalid");

    }
}
```

Q61: **What is the difference between Array and ArrayList? Write a program to concatenate str1 and str2.**
Ans:

- Array → fixed size, can store primitives & objects.
- ArrayList → dynamic size, stores only objects.

```
class Concat {

  public static void main(String[] args){

    String s1="Hello", s2="World";

    System.out.println(s1+s2);

  }

}
```

Q62: **How can we check if an array contains values or not? Print sum of squares of first two elements.**
Ans:

- Check → if(arr.length==0) → empty.

```
class SumSquares {

  public static void main(String[] args){

    int[] arr={3,4,5};

    if(arr.length>=2){

      int sum=arr[0]*arr[0]+arr[1]*arr[1];

      System.out.println(sum);

    }

  }

}
```

Q63: **Advantages and disadvantages of array? Differences between array and ArrayList.**
Ans:

- Advantages: Fast access, simple structure.
- Disadvantages: Fixed size, cannot grow/shrink.
- Array vs ArrayList → Array fixed, ArrayList resizable.

Q64: **What is static variable? What is static method?**
Ans:

- **Static variable** → shared among all objects of a class.
- **Static method** → belongs to class, can be called without object.

Q65: **Write a program to calculate factorial of a given number.**
Ans:

```
class Factorial {

    public static void main(String[] args){

        int n=5,f=1;

        for(int i=1;i<=n;i++) f*=i;

        System.out.println(f);

    }

}
```

Q66: **How can we run a java program without making any object? How to sort ArrayList?**
Ans: By using static methods & main().
Sort → Collections.sort(list);

Q67: **Similarity and Difference between static block and static method?**
Ans:

- Similarity: Both belong to class, not object.
- Difference: Static block runs once during class loading, static method is called explicitly.

Q68: **How can we create objects if we make the constructor private?**
Ans: By using **factory method** inside the class (Singleton pattern).

Q69: **Is it possible to compile and run a Java program without writing main() method?**
Ans: Compile → Yes, Run → No. JVM needs main() as entry point.

Q70: **Can we initialize member variables within static block?**
Ans: Yes, but only **static variables** can be initialized inside static block.

Q71: **What is the purpose of this keyword? Write program to implement Binary Search.**
Ans: this refers to current object.

```
class BinarySearch {

    static int search(int arr[],int key){

        int l=0,r=arr.length-1;

        while(l<=r){

            int mid=(l+r)/2;

            if(arr[mid]==key) return mid;

            if(arr[mid]<key) l=mid+1; else r=mid-1;
        }

        return -1;

    }

    public static void main(String[] args){

        int arr[]={10,20,30,40,50};

        System.out.println(search(arr,30));

    }

}
```

Q72: **How do you implement inheritance practically in Java?**

Ans: Using extends keyword.

```
class A { void show(){System.out.println("A");} }

class B extends A { void display(){System.out.println("B");} }
```

Q73: **What is the purpose of inheritance?**

Ans: To achieve **code reusability** and establish parent-child relationship.

Q74: **What are generalized and specialized classes in Java?**

Ans:

- **Generalized class** → parent/base class with common features.
- **Specialized class** → child class with additional/specific features.

Q75: **What are the types of inheritance?**

Ans:

- Single, Multilevel, Hierarchical, Hybrid (via interfaces).
  (Multiple inheritance not supported with classes, only interfaces).

Q76: What is the purpose of the 'super' keyword in Java?

Ans: super is used to refer to the immediate parent class. It can access parent variables, methods, and constructors.

Q77: What are the differences between 'this' and 'super' keyword?

Ans:

- this → Refers to current object.
- super → Refers to parent class object.
- this() → Calls current class constructor.
- super() → Calls parent class constructor.

Q78: What are the rules to be followed while overriding a method?

Ans:

1. Same method name and parameters.
2. Return type must be same or covariant.
3. Cannot reduce visibility (public → private not allowed).

4. Cannot override final or static methods.

Q79: What is the base class of all classes? Explain multiple inheritance with an example.

Ans: The base class is **Object**.

Java does not support multiple inheritance with classes (to avoid ambiguity), but supports it via **interfaces**.

Q80: Does Java support multiple inheritances? Does it exist in Java?

Ans: Java does **not** support multiple inheritance with classes but it exists using **interfaces**.

Q81: How to define a constant variable in Java?

Ans: Use final keyword. Example:

```
final int MAX = 100;
```

Q82: What is the purpose of declaring a variable as 'final'? What is the impact of declaring a method as final?

Ans:

- Final variable → Value cannot change.
- Final method → Cannot be overridden.

Q83: Explain call by value and call by reference with example.

Ans:

- Java uses **call by value** (copies the value).
- For objects, the reference is copied (changes inside method affect object).

Q84: I don't want my class to be inherited by any other class. What should I do?

Ans: Declare the class as **final**.

Q85: Can you give a few examples of final classes defined in Java API?

Ans: String, Integer, Math, System.

Q86: How is final different from finally and finalize()?

Ans:

- final → Keyword (constant, prevent override/inherit).
- finally → Block in exception handling (always executes).
- finalize() → Method called before object garbage collection.

Q87: Does a class inherit the constructors of its superclass?

Ans: No, constructors are **not inherited**, but subclass can call parent constructor using super().

Q88: What is Overriding?

Ans: Overriding is when a subclass provides its own implementation of a method already defined in its superclass.

Q89: How is this() and super() used with constructors?

Ans:

- this() → Calls another constructor in the same class.
- super() → Calls parent class constructor.
  Both must be the **first statement** in constructor.

Q90: What modifiers are allowed for methods in an Interface?

Ans:

- By default → public abstract.
- Since Java 8 → default and static.
- Since Java 9 → private (helper methods).

Q91: What are some alternatives to inheritance?

Ans:

- **Composition** (has-a relationship)
- **Interfaces / Abstract classes**
- **Delegation**

Q92: Does a class inherit the constructors of its superclass?

Ans: No. Constructors are **not inherited**, but can be called using super().

Q93: What restrictions are placed on method overloading?

Ans:

- Must have same method name.
- Parameter lists must differ.
- Return type can differ but does **not** distinguish overloaded methods.
- Cannot differ only by throws clause.

Q94: What is method overloading & method overriding?

Ans:

- **Overloading** → Same name, different parameters, same class (compile-time).
- **Overriding** → Subclass redefines parent method, same signature (runtime).

Q95: What is the difference between overloading & overriding?

Ans:

Feature	Overloading	Overriding
Parameters	Must differ	Same
Class	Same class	Subclass
Return Type	Can differ	Same/covariant
Binding	Compile-time	Runtime

Q96: What is the difference between the superclass and the subclass?

Ans:

- **Superclass** → Parent/base class.
- **Subclass** → Child class that inherits from superclass, can add/override features.

Q97: What modifiers may be used with top-level class?

Ans: public or default (package-private). Cannot be private or protected.

Q98: What do you understand by an unreachable catch block error?

Ans: A catch block is unreachable if its exception type is already handled by a previous catch or is impossible (e.g., subclass after superclass).

Q99: Can a class extend itself?

Ans: No, a class cannot extend itself (causes **cyclic inheritance** error).

Q100: What is order of calling constructors in case of inheritance?

Ans: Parent (superclass) constructor is called **first**, then subclass constructor.

Q101: What happens if both superclass and subclass have a field with the same name?

Ans: Field in subclass **hides** the superclass field. Access:

- subObj.field → subclass field
- super.field → superclass field

Q102: Can we access both superclass and subclass members if we create an object of subclass?

Ans: Yes, subclass object can access subclass members directly and superclass members via inheritance or super.

Q103: Which of the following is correct way of inheriting class A by class B?

Q104: class B + class A { }

Ans: ❌ Incorrect syntax

Q105: class B inherits class A { }

Ans: ❌ Incorrect. Correct → class B extends A { }

Q106: class B extends A { }

Ans: ✅ Correct way to inherit class A.

Q107: class B extends class A { }

Ans: ❌ Incorrect syntax, class keyword is not repeated.

Q108: Are static members inherited to subclasses?

Ans: Yes, static members are inherited but belong to the **class**, not the object.

Q109: What are abstract classes? Properties? Example?

Ans:

- **Abstract class** → cannot be instantiated, can have abstract & concrete methods.
- **Properties:**
  - Can have abstract and non-abstract methods
  - Can have constructors
  - Can have member variables
- **Example:** abstract class Vehicle { abstract void start(); } → subclasses implement start().

Q110: Private vs Protected vs Public vs Default access

Ans:

Access	Class	Package	Subclass	World
private	✅	❌	❌	❌
default	✅	✅	❌	❌
protected	✅	✅	✅	❌
public	✅	✅	✅	✅

Q111: What is Object Wrapper and Autoboxing in Java?

Ans:

- **Wrapper** → Object representation of primitive types (Integer, Double).
- **Autoboxing** → Automatic conversion between primitive and wrapper (int → Integer).

Q112: What is Object Class in Java?

Ans: Object is the **root class** of all Java classes; provides common methods like toString(), equals(), hashCode().

Q113: Difference between compile-time and runtime polymorphism

Ans:

- **Compile-time** → Method overloading (decided at compile time)
- **Runtime** → Method overriding (decided at runtime)

Q114: What is Runtime Polymorphism?

Ans: Subclass method **overrides** superclass method; object reference determines method called at runtime.

Q115: Can you achieve Runtime Polymorphism by data members?

Ans: ❌ No, runtime polymorphism works only for **methods**, not fields.

Q116: Difference between static binding and dynamic binding

Ans:

- **Static binding** → Compile-time (method overloading, private/final methods)
- **Dynamic binding** → Runtime (overridden methods)

Q117: What is Java instanceof operator?

Ans: Checks whether an object is an instance of a specific class or subclass.

```
if(obj instanceof String){...}
```

Q118: Difference between abstraction and encapsulation

Ans:

- **Abstraction** → Hides implementation, shows behavior (abstract class/interface).
- **Encapsulation** → Hides data using private variables + getters/setters.

Q119: What is an abstract class?

Ans: Class that **cannot be instantiated** and may contain **abstract methods**.

Q120: Can there be an abstract method without an abstract class?

Ans: ❌ No, abstract methods must be inside an abstract class.

Q121: Can you use abstract and final both with a method?

Ans: ❌ No, abstract method cannot be final because abstract methods must be overridden.

Q122: Is it possible to instantiate an abstract class?

Ans: ❌ No, abstract classes **cannot be instantiated** directly.

Q123: What is an interface?

Ans: A **blueprint of a class**; contains abstract methods (Java 8+ can have default & static methods) and constants.

Q124: Can you declare an interface method static?

Ans: ✅ Yes, from Java 8 onwards, interface can have **static methods**.

Q125: Can the Interface be final?

Ans: ❌ No, interface cannot be final because it is meant to be implemented.

Q126: What is a marker interface?

Ans: Interface with **no methods**; used to mark a class for special behavior (e.g., Serializable).

Q127: Can we define private and protected modifiers for members in interfaces?

Ans:

- **Fields** → always public static final
- **Methods** → can be private (Java 9+) for internal use; **protected** not allowed

Q128: When can an object reference be cast to an interface reference?

Ans: When the object **implements the interface**, you can cast:

Interface i = (Interface) obj;

Q129: How to make a read-only class in Java?

Ans:

- Declare fields **private + final**
- Provide **only getters**, no setters

Q130: How to make a write-only class in Java?

Ans:

- Declare fields **private**
- Provide **only setters**, no getters

Q131: What is a package?

Ans: A **namespace** for organizing classes and interfaces in Java.

Q132: Advantages of defining packages in Java

Ans:

- Avoid class name conflicts
- Easier maintenance
- Access protection
- Reusability

Q133: How to create packages in Java?

Ans:

package myPackage; // top of Java file

Compile: javac -d . MyClass.java

Q134: How can we access a class in another package?

Ans: Use **import** statement:

import packageName.ClassName;

Q135: Do I need to import java.lang package? Why?

Ans: ❌ No, java.lang is **imported by default**.

Q136: Can I import same package/class twice? Will the JVM load the package twice at runtime?

Ans: ✅ You can import multiple times, but JVM **loads a class only once**.

Q137: What is the static import?

Ans: Allows accessing **static members** (fields/methods) **without class name**:

import static java.lang.Math.*;

int x = sqrt(16);

Q138: How many types of exception can occur in a Java program?

Ans: Two types: **Checked Exceptions** and **Unchecked Exceptions** (Runtime exceptions).

Q139: What is Exception Handling?

Ans: Mechanism to **handle runtime errors** using try, catch, finally, throw, throws.

Q140: Explain the hierarchy of Java Exception classes

Ans:

```
Throwable

 ├── Error

 └── Exception

     ├── Checked Exception

     └── RuntimeException (Unchecked)
```

Q141: Difference between Checked and Unchecked Exception

Ans:

- **Checked** → Checked at **compile-time** (IOException)
- **Unchecked** → Checked at **runtime** (NullPointerException)

Q142: Base class for Error and Exception

Ans: Throwable

Q143: Is it necessary that each try block must be followed by a catch block?

Ans: ❌ No, try block can be followed by **finally** without catch.

Q144: What is finally block?

Ans: Block that **always executes**, used for cleanup (e.g., closing resources).

Q145: Can finally block be used without a catch?

Ans: ✅ Yes:

try { /* code */ } finally { /* cleanup */ }

Q146: Is there any case when finally will not be executed?

Ans: Yes, if JVM **crashes** or System.exit() is called before finally.

Q147: Difference between throw and throws

Ans:

- throw → **used to throw a single exception**
- throws → **declares exception(s) in method signature**

Q148: Can an exception be rethrown?

Ans: ✅ Yes, using throw inside a catch block.

Q149: Can subclass overriding method declare an exception if parent class method doesn't throw an exception?

Ans: ❌ No, subclass **cannot throw new checked exceptions** not declared in parent method.

Q150: What is exception propagation?

Ans: When an exception is **passed up the call stack** until handled by a suitable catch.

Q151: Can we have statements between try, catch and finally blocks?

Ans: ❌ No, **try, catch, finally must follow each other immediately**.

Q152: What is Exception Chaining?

Ans: Wrapping a **new exception** around a **caught exception** to provide more context using Throwable.initCause().

Q153: Can you catch and handle Multiple Exceptions in Java?

Ans: ✅ Yes, using **multi-catch**:

catch(IOException | SQLException e) { /* handle */ }

Q154: Differentiate between Checked Exception and Unchecked Exceptions in Java

Ans:

- **Checked** → Compile-time, must be handled (IOException)
- **Unchecked** → Runtime, optional to handle (NullPointerException)

Q155: How do you handle checked exceptions?

Ans: Using **try-catch block** or **throws** declaration in method signature.

Q156: What are runtime exceptions in Java?

Ans: Exceptions **occurring at runtime**, subclass of RuntimeException (e.g., ArithmeticException, NullPointerException).

Q157: What are the important methods defined in Java's Exception Class?

Ans:

- getMessage() → Returns error message
- printStackTrace() → Prints call stack
- toString() → Returns exception info

Q158: How are exceptions handled in Java?

Ans: Using **try-catch-finally blocks**, **throw**, and **throws** keywords.

Q159: Best practices in Java Exception Handling

Ans:

- Catch **specific exceptions**
- Avoid empty catch blocks
- Use **finally** to release resources
- Prefer **checked exceptions** for recoverable errors
- Don't use exceptions for normal flow

Q160: Rules when overriding a method throwing an Exception

Ans:

- Subclass method can **throw same, subclass, or no exception**
- Cannot throw **new checked exceptions** not in parent

Q161: Are we allowed to use only try blocks without a catch and finally blocks?

Ans: ❌ No, try must have either **catch or finally**.

Q162: Does finally block always get executed in Java?

Ans: ✅ Generally yes, except **if JVM exits** or **thread is killed**.

Q163: Under what circumstances should we subclass an Exception?

Ans: To create **custom exceptions** for **application-specific error handling**.

Q164: Scenarios where "Exception in thread main" could occur

Ans: Unhandled **runtime exception** in main method, e.g., NullPointerException, ArrayIndexOutOfBoundsException.

Q165: What happens to the exception object after exception handling is complete?

Ans: It becomes **eligible for garbage collection** if no references remain.

Q166: Is it possible to throw checked exceptions from a static block?

Ans: ❌ No, static blocks **cannot throw checked exceptions** directly. They must handle them with **try-catch**.

Q167: What happens when an exception is thrown by the main method?

Ans: If unhandled, the JVM **prints stack trace and terminates the program**.

Q168: What does JVM do when an exception occurs in a program?

Ans: JVM searches **catch block** to handle it; if not found, **propagates up** and terminates the thread.

Q169: Explain Java Exception Hierarchy.

Ans:

- Throwable
  - Error → JVM errors (OutOfMemoryError)
  - Exception
    - Checked → Compile-time (IOException)
    - Unchecked → Runtime (NullPointerException)

Q170: What is a nested class?

Ans: A class **defined within another class**.

Q171: What are the advantages of Java inner classes?

Ans:

- Can access **outer class members**
- Improves **encapsulation**
- Logical grouping of classes

Q172: What are the disadvantages of using inner classes?

Ans:

- Increases **code complexity**
- Slight **performance overhead**

Q173: Types of inner classes (non-static nested class) in Java

Ans:

- **Member inner class**
- **Local inner class**
- **Anonymous inner class**

Q174: What are anonymous inner classes?

Ans: Classes **without a name**, declared and instantiated **in a single statement**.

Q175: What is the nested interface?

Ans: An **interface declared within a class**.

Q176: Can a class have an interface?

Ans: ✅ Yes, a class can **implement an interface**.

Q177: Can an Interface have a class?

Ans: ✅ Yes, an interface can have **nested static class**.

Q178: What is Garbage Collection?

Ans: Automatic **reclaiming of memory** of objects no longer referenced.

Q179: What is gc()?

Ans: Method in **Runtime class** requesting JVM to **perform garbage collection**.

Q180: How is garbage collection controlled?

Ans: Cannot be forced; JVM controls it, but we can **suggest using System.gc() or Runtime.gc()**.

Q181: How can an object be unreferenced? What is the purpose of the Runtime class?

Ans:

- **Unreferenced** → when no variable points to the object
- **Runtime class** → Provides **JVM runtime info** and **methods to interact** (e.g., memory, GC, exec processes)

Q182: Differences between Runtime and Compile-time Polymorphism

Ans:

- **Runtime (Dynamic)** → Method overriding, resolved at runtime
- **Compile-time (Static)** → Method overloading, resolved at compile time

Q183: How will you invoke any external process in Java?

Ans: Using **Runtime.getRuntime().exec("command")** or ProcessBuilder class.

Q184: What are wrapper classes?

Ans: Classes that **wrap primitive types** as objects, e.g., Integer for int, Double for double.

Q185: What is a thread?

Ans: Lightweight unit of execution within a process; allows **concurrent execution**.

Q186: Differentiate between process and thread?

Ans:

- **Process:** Independent program execution, has its **own memory**
- **Thread:** Lightweight unit within a process, shares **process memory**

Q187: What do you understand by inter-thread communication?

Ans: Mechanism allowing **threads to communicate** and **coordinate** execution using **wait(), notify(), notifyAll()**.

Q188: What is the purpose of wait() method in Java?

Ans: Causes the thread to **wait until another thread invokes notify() or notifyAll()** on the same object.

Q189: Why must wait() method be called from the synchronized block?

Ans: Because wait() **releases the object lock**, so it must own the lock to wait safely.

Q190: What are the advantages of multithreading?

Ans:

- Better **CPU utilization**
- **Concurrent execution**
- Faster **program execution**
- Improved **responsiveness**

Q191: What are the states in the lifecycle of a Thread?

Ans:

- **New** → Created but not started
- **Runnable** → Ready to run
- **Running** → Executing
- **Waiting/Blocked** → Waiting for resources or notify
- **Timed Waiting** → Waiting for specific time
- **Terminated** → Completed execution

Q192: What is the difference between preemptive scheduling and time slicing?

Ans:

- **Preemptive:** Higher priority thread runs; lower waits
- **Time slicing:** CPU shares time slices among threads **round-robin**

Q193: What is context switching?

Ans: Saving the state of a thread/process and **switching CPU to another thread/process**.

Q194: Differentiate between the Thread class and Runnable interface for creating a Thread

Ans:

- **Thread class:** Extend Thread class, single inheritance limit
- **Runnable interface:** Implement Runnable, can extend other classes

Q195: What does join() method do?

Ans: Makes **current thread wait** until the target thread **finishes execution**.

Q196: Describe the purpose and working of sleep() method.

Ans: Pauses a thread for **specific milliseconds**, **does not release lock**.

Q197: Discuss about the life cycle of a Thread.

Ans: New → Runnable → Running → Waiting/Blocked/Timed Waiting → Terminated

Q198: What is the difference between wait() and sleep() method?

Ans:

- **wait():** Releases lock, used for **thread communication**
- **sleep():** Does not release lock, just **pauses thread**

Q199: Is it possible to start a thread twice?

Ans: ❌ No, start() can be called **only once per thread**.

Q200: Can we call the run() method instead of start()?

Ans: Yes, but it **executes in the current thread**, not a new thread.

Q201: What is shutdown hook?

Ans: Thread that executes when **JVM shuts down**, e.g., cleanup activities.

Q202: When should we interrupt a thread?

Ans: When we want to **stop a thread gracefully** or wake it from **sleep/wait**.

Q203: What is synchronization?

Ans: Mechanism to **control access** of multiple threads to shared resources.

Q204: What is the purpose of the Synchronized block?

Ans: Allows **locking only a specific code block**, not the whole method, improving efficiency.

Q205: What is the difference between notify() and notifyAll()?

Ans:

- **notify():** Wakes **one waiting thread**
- **notifyAll():** Wakes **all waiting threads**

Q206: What is Thread Scheduler in Java?

Ans: Part of JVM that **decides which thread runs** and **for how long**, based on priority and thread state.

Q207: Does each thread have its stack in multithreaded programming?

Ans: ✅ Yes, each thread has its **own stack memory** for local variables and method calls.

Q208: What is race-condition?

Ans: Situation where **two or more threads access shared data** simultaneously, causing **unexpected results**.

Q209: What is the volatile keyword in Java?

Ans: Ensures **visibility of changes** to a variable across threads, **prevents caching** in CPU registers.

Q210: What do you understand by thread pool?

Ans: Collection of **pre-created reusable threads** used to **execute tasks efficiently**, reducing thread creation overhead.

Q211: What is the difference between synchronous and asynchronous programming?

Ans:

- **Synchronous:** Tasks execute **one after another**
- **Asynchronous:** Tasks execute **independently**, without waiting

Q212: Between synchronized method and synchronized block, which do you prefer?

Ans: Synchronized block – more efficient, locks only **critical section**, not the entire method.

Q213: What is a thread scheduler, and how is it related to thread priority?

Ans: Thread scheduler **allocates CPU time** based on **thread priority**, higher priority threads get preference.

Q214: What is time slicing?

Ans: CPU **divides time** among threads **round-robin**, giving each thread a small time quantum.

Q215: What is the busy spin technique?

Ans: Thread **continuously checks** for a condition without sleeping, **wasting CPU cycles**.

Q216: What is thread starvation?

Ans: Thread cannot **get CPU time** for long duration due to **low priority or other threads consuming resources**.

Q217: Why are the methods notify(), wait() and notifyAll() in Object class?

Ans: Every object can be a **monitor**, so these methods belong to **Object**, not Thread.

Q218: How are the methods wait() and sleep() different?

Ans:

- **wait():** Releases lock, used for **thread communication**
- **sleep():** Does not release lock, just **pauses thread**

Q219: Why is it important to override the run() method in a thread class?

Ans: run() contains the **code executed by the thread**; without it, thread does nothing.

Q220: Explain how you would stop the execution of a thread in Java.

Ans: Use **interrupt()** or a **boolean flag** to terminate gracefully; avoid using deprecated stop().

Q221: What is the difference between sleep() and suspend()?

Ans:

- **sleep():** Pauses thread **temporarily**, continues after time
- **suspend():** Pauses thread **indefinitely**, can cause **deadlock** (deprecated)

Q222: What are some fundamental advantages of multithreading in Java?

Ans:

- Improved **CPU utilization**
- **Concurrent execution**
- Better **responsiveness**
- Efficient **resource sharing**

Q223: What are some functions used to perform inter-thread communication in Java?

Ans: wait(), notify(), notifyAll()