**OOPS AND CORE JAVA:**

**🔹 1. What are the four main pillars of OOPs in Java?**

**Answer:**

1. **Encapsulation – Wrapping data (variables) and methods into a single unit (class).**
2. **Abstraction – Hiding complex internal details and showing only essential features.**
3. **Inheritance – One class (child) inherits properties and behavior from another class (parent).**
4. **Polymorphism – One thing behaves differently in different contexts (method overloading & overriding).**

**2. What is Encapsulation? How do you implement it in Java?**

**Answer:** Encapsulation is the practice of keeping fields private and providing public getter/setter methods to access and modify them.

class Student {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**🔹 3. What is Inheritance? Types of Inheritance in Java?**

**Answer:** Inheritance allows a class to acquire properties and methods of another class using the extends keyword.

Types supported in Java:

* Single
* Multilevel
* Hierarchical

**Note:** Java doesn't support multiple inheritance using classes (only through interfaces).

**🔹 4. What is Polymorphism? Types of Polymorphism?**

**Answer:** Polymorphism means "many forms."

**Types:**

* **Compile-time Polymorphism** (Method Overloading)
* **Runtime Polymorphism** (Method Overriding)

**🔹 5. What is Abstraction? How is it implemented?**

**Answer:** Abstraction hides implementation details and shows only functionality to the user.

**Implemented using:**

* **Abstract classes**
* **Interfaces**

abstract class Shape {

abstract void draw();

}

class Circle extends Shape {

void draw() {

System.out.println("Drawing Circle");

}

}

**🔹 6. What is the difference between Abstraction and Encapsulation?**

| **Abstraction** | **Encapsulation** |
| --- | --- |
| Hides implementation | Hides data |
| Achieved using abstract class interfaces | Achieved using access modifiers (private, public) |
| Focuses on what to do | Focuses on how to do it |

**7. What is the difference between Overloading and Overriding?**

| **Overloading** | **Overriding** |
| --- | --- |
| Compile-time | Runtime |
| Same method name, different parameters | Same name and parameters |
| Happens in same class | Happens in subclass |

**🔹 8. What is the difference between a class and an object?**

* **Class**: Blueprint or template (e.g., Car).
* **Object**: Instance of the class (e.g., myCar).

**🔹 9. What is a constructor?**

A constructor is a special method used to initialize objects. It has the same name as the class and no return type.

**🔹 10. What is this and super keyword?**

* this → Refers to current class instance.
* super → Refers to parent class instance or constructor.

**11. What is the difference between == and .equals()?**

* == compares **object references**.
* .equals() compares **object content** (can be overridden).

**🔹 12. What is the difference between final, finally, and finalize()?**

| **Keyword** | **Purpose** |
| --- | --- |
| final | Prevent modification (class, method, variable) |
| finally | Block always executed after try-catch |
| finalize() | Called by GC before destroying object |

**EXCEPTION HANDING :**

**13. What is an Exception in Java?**

**Answer:** An exception is an unwanted or unexpected event that disrupts the normal flow of a program.

**🔹 14. What is the difference between Error and Exception?**

| **Feature** | **Exception** | **Error** |
| --- | --- | --- |
| Recoverable | Yes | Usually No |
| Occurrence | Handled by try-catch | Occurs due to system issues |
| Example | NullPointerException, IOException | OutOfMemoryError, StackOverflowError |

**🔹 15. What is the difference between Checked and Unchecked Exception?**

| **Type** | **Checked Exception** | **Unchecked Exception** |
| --- | --- | --- |
| Checked at | Compile-time | Runtime |
| Needs try-catch? | Yes | Optional |
| Examples | IOException, SQLException | NullPointerException, ArithmeticException |

**🔹 16. What are the keywords used in Exception Handling?**

* try: Wraps risky code.
* catch: Handles the exception.
* finally: Always executed (cleanup code).
* throw: Used to explicitly throw an exception.
* throws: Declares exceptions.

**🔹 17. What is the difference between throw and throws?**

| **throw** | **throws** |
| --- | --- |
| Used to throw exception manually | Declares that method may throw exception |
| Used inside method | Used with method signature |
| Example: throw new IOException() | Example: public void readFile() throws IOException |

**🔹 18. What is the purpose of the finally block?**

**Answer:** It is used to execute important code like resource cleanup. It always executes regardless of whether an exception occurs or not.

try {

// risky code

} catch (Exception e) {

// handle

} finally {

// cleanup code (e.g., closing file, DB)

}

**🔹 19. Can we have try block without catch or finally?**

**Answer:** No. A try block must be followed by either catch or finally or both.

**🔹 20. What is a custom exception?**

**Answer:** An exception created by the user to handle specific business logic.

class AgeException extends Exception {

AgeException(String msg) {

super(msg);

}

}

**🔹 21. What is the difference between multiple catch and multi-catch block (Java 7+)?**

**Multiple catch:**

try {

} catch(IOException e) {

} catch(SQLException e) {

}

**Multi-catch (Java 7+):**

try {} catch(IOException | SQLException e) {}

**🔹 22. What is try-with-resources?**

**Answer:** Introduced in Java 7 — used to automatically close resources like files, DB connections.

java

try (BufferedReader br = new BufferedReader(new FileReader("file.txt"))) {

// use br

}

**STRING CONCEPT:**

**23. What is a String in Java?**

**Answer:**  
A String is an object in Java that represents a sequence of characters. It's immutable, meaning once created, its value cannot be changed.

String str = "Hello";

**🔹 24. Why are Strings immutable in Java?**

**Answer:**

* **Security**: Used in network connections, file paths, etc.
* **Thread safety**: No synchronization required.
* **Caching**: Can be reused from the String pool.
* **Performance**: Enables string pooling and better memory efficiency.

**🔹 25. What is the String Pool in Java?**

**Answer:**  
A special memory area in the heap where Java stores string literals. If a string already exists in the pool, the same reference is returned.

String a = "Java";

String b = "Java";

System.out.println(a == b); // true

**🔹 25 Difference between == and equals() for String comparison?**

| **==** | **equals()** |
| --- | --- |
| Compares references | Compares values (content) |
| Checks if same object | Checks if same characters |
| Returns true if pointing to same memory | Returns true if same content |

**🔹 26. How to create a mutable String?**

**Answer:**  
Use StringBuilder or StringBuffer for mutable strings.

StringBuilder sb = new StringBuilder("Hello");

sb.append(" World");

System.out.println(sb); // Hello World

**🔹 27. What is the difference between StringBuilder and StringBuffer?**

| **StringBuilder** | **StringBuffer** |
| --- | --- |
| Not thread-safe | Thread-safe (synchronized) |
| Faster | Slower |
| Preferred in single-thread | Preferred in multi-thread |

**🔹 28. Commonly used String methods?**

* length()
* charAt(index)
* substring()
* indexOf()
* lastIndexOf()
* equals()
* equalsIgnoreCase()
* toLowerCase()
* toUpperCase()
* replace()
* split()
* trim()
* startsWith() / endsWith()

**🔹 29. Difference between substring() and split()?**

* substring(int start, int end) – Extracts part of the string.
* split(String regex) – Splits string into an array using regex.

**🔹 30. What is the output of this code?**

String a = "Hello";

a.concat(" World");

System.out.println(a);

**Answer:** Hello  
Because strings are immutable. The concat() method returns a new string, but the result wasn't assigned.

**🔹 31. How to reverse a String in Java?**

**Using StringBuilder:**

String str = "Java";

String reversed = new StringBuilder(str).reverse().toString();

System.out.println(reversed);

**MULTI THREADING :**

**32. What is a thread in Java?**

**Answer:**  
A thread is the smallest unit of a process. In Java, it's a lightweight subprocess that enables multitasking (doing multiple things simultaneously).

**🔹 33. How do you create a thread in Java?**

**Two ways:**

1. **Extend Thread class:**

class MyThread extends Thread {

public void run() {

System.out.println("Running...");

}

}

1. **Implement Runnable interface:**

class MyRunnable implements Runnable {

public void run() {

System.out.println("Running...");

}

}

**🔹 34. What is the difference between start() and run() method?**

| **start()** | **run()** |
| --- | --- |
| Starts a new thread | Runs on the current thread |
| Calls run() internally | Just a normal method call |
| Multithreading enabled | No multithreading |

**🔹 35. What is thread lifecycle in Java?**

**States:**

* New
* Runnable
* Running
* Waiting
* Timed Waiting
* Terminated

**🔹 36. What is synchronization in Java?**

**Answer:**  
Used to control access to shared resources by multiple threads. It prevents **race conditions**.

synchronized void print() {

// synchronized code

}

**🔹 37. What is a deadlock?**

**Answer:**  
Deadlock occurs when two or more threads are blocked forever, each waiting for the other to release a lock.

**🔹 38. What are wait(), notify(), and notifyAll()?**

| **Method** | **Description** |
| --- | --- |
| wait() | Causes current thread to wait and release the lock |
| notify() | Wakes up a single waiting thread |
| notifyAll() | Wakes up all waiting threads |

They must be used inside a synchronized block.

**🔹 39. Difference between sleep() and wait()?**

| **Feature** | **sleep()** | **wait()** |
| --- | --- | --- |
| From class | Thread class | Object class |
| Lock released? | No | Yes |
| Used for | Pause thread | Thread communication |

**🔹 40. What is volatile keyword?**

**Answer:**  
It ensures the latest value of a variable is always read from main memory (not cached). It's used in multi-threaded environments for visibility.

**🔹 41. What is ExecutorService?**

**Answer:**  
It is part of the concurrency API to manage a pool of threads (instead of manually creating threads).

ExecutorService service = Executors.newFixedThreadPool(5);

service.submit(() -> System.out.println("Task"));

service.shutdown();

**🔹 42. What is the difference between Thread and Runnable?**

| **Feature** | **Thread Class** | **Runnable Interface** |
| --- | --- | --- |
| Inheritance | Can’t extend other class | Can implement multiple interfaces |
| Resource sharing | Difficult | Easy |
| Preferred | For simple thread logic | For large-scale thread management |

**🔹 43. What is Callable and Future?**

**Answer:**  
Used to return results from threads.

Callable<String> task = () -> "Result";

Future<String> result = executor.submit(task);

System.out.println(result.get()); // blocking call