⭐ 30 Core Java Important Questions with Full Pattern

Here are **only questions** (for index purpose):

1. What is Abstraction?
2. What is the difference between Abstract Class and Interface?
3. What is Functional Interface?
4. What is Marker Interface?
5. What is the internal working of HashMap?
6. What is Serialization and Deserialization?
7. What is Synchronization?
8. What is Encapsulation?
9. What is Inheritance?
10. What is Polymorphism?
11. What is Method Overloading?
12. What is Method Overriding?
13. What is Constructor in Java?
14. What is the this keyword?
15. What is the super keyword?
16. What is the static keyword?
17. What is the final keyword?
18. What is Exception Handling?
19. What is the difference between Checked and Unchecked Exceptions?
20. What is Multithreading?
21. What is Runnable Interface?
22. What is String Pool in Java?
23. What is the difference between String, StringBuilder, and StringBuffer?
24. What is the Collection Framework in Java?
25. What is the difference between List and Set?
26. What is the difference between ArrayList and LinkedList?
27. What is Comparable Interface?
28. What is Comparator Interface?
29. What is Stream API in Java?
30. What is Lambda Expression?

🔹 1. What is Abstraction?

📘 Detailed Definition

Abstraction is the concept of hiding implementation details and showing only essential functionality to the user. It focuses on *what object does* instead of *how it does it*.

📝 Short Answer

Abstraction hides internal implementation and exposes only required features.

💻 Example

Java

```
abstract class Vehicle {
    abstract void start();

    void fuel() {
        System.out.println("Vehicle needs fuel");
    }
}

class Car extends Vehicle {
    void start() {
        System.out.println("Car starts with key");
    }
}

public class Test {
    public static void main(String[] args) {
        Vehicle v = new Car();
        v.start();
        v.fuel();
    }
}
```

⭐ Why / Where / When Used

- Reduces complexity
- Used in large systems
- Used when implementation should be hidden

🔹 2. Difference Between Abstract Class and Interface

📘 Detailed Definition

Abstract class can have abstract and concrete methods. Interface contains only method declarations (except default/static methods).

📝 Short Answer

Abstract class = Partial abstraction
Interface = Full abstraction (mostly)

💻 Example

Java

```
abstract class Animal {
    abstract void sound();
}

interface Flyable {
    void fly();
}

class Bird extends Animal implements Flyable {
    void sound() {
        System.out.println("Bird chirps");
    }

    public void fly() {
        System.out.println("Bird flies");
    }
}
```

⭐ Why / Where / When Used

- Abstract class → When sharing common code
- Interface → When multiple capabilities required

🔹 3. What is Functional Interface?

📘 Detailed Definition

An interface having only one abstract method is called Functional Interface.

📝 Short Answer

Single abstract method interface.

💻 Example

```java
@FunctionalInterface
interface Calculator {
    int add(int a, int b);
}

public class Test {
    public static void main(String[] args) {
        Calculator c = (a, b) -> a + b;
        System.out.println(c.add(5, 3));
    }
}
```

⭐ Why / Where / When Used

- Used in Lambda Expressions
- Used in Stream API
- Used in Event Handling

🔹 4. What is Marker Interface?

📘 Detailed Definition

Marker interface is an empty interface used to provide special instructions to JVM.

📝 Short Answer

Interface without methods.

💻 Example

```java
import java.io.Serializable;

class Student implements Serializable {
    int id = 10;
}
```

⭐ Why / Where / When Used

- Enables special behavior
- Example → Serializable, Cloneable

🔹 5. Internal Working of HashMap

📘 Detailed Definition

HashMap stores data in key-value pairs using hashing technique. It converts key into hashcode and stores data in bucket.

📝 Short Answer

HashMap uses hashing and bucket storage.

💻 Example

```java
import java.util.*;
```

```
public class Test {
    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<>();
        map.put(1, "Java");
        map.put(2, "Spring");

        System.out.println(map.get(1));
    }
}
```

⭐ Why / Where / When Used

- Fast data retrieval
- Used in caching
- Used in configuration storage

🔹 6. What is Serialization and Deserialization?

📘 Detailed Definition

Serialization converts object into byte stream. Deserialization converts byte stream back to object.

📝 Short Answer

Object → Byte Stream → Object

💻 Example

Java

```
import java.io.*;

class Student implements Serializable {
    int id = 1;
}
```

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

        Student s = new Student();

        ObjectOutputStream oos =
            new ObjectOutputStream(new
FileOutputStream("file.txt"));
        oos.writeObject(s);

        ObjectInputStream ois =
            new ObjectInputStream(new
FileInputStream("file.txt"));
        Student s2 = (Student) ois.readObject();

        System.out.println(s2.id);
    }
}
```

⭐ Why / Where / When Used

- Used in file storage
- Used in network communication
- Used in session storage

🔹 7. What is Synchronization?

📘 Detailed Definition

Synchronization controls multiple threads accessing shared resources to prevent data inconsistency.

📝 Short Answer

Thread control mechanism.

💻 Example

```java
class Counter {
    int count = 0;

    synchronized void increment() {
        count++;
    }
}
```

⭐ Why / Where / When Used

- Prevent race condition
- Used in multi-threading

🔹 8. What is Encapsulation?

📘 Definition

Binding data and methods into single unit.

📝 Short Answer

Data hiding using getters and setters.

💻 Example

```java
class Student {
    private int id;

    public void setId(int id) {
        this.id = id;
    }

    public int getId() {
```

```
        return id;
    }
}
```

⭐ Why / Where / When Used

- Security
- Data control

🔹 9. What is Inheritance?

📘 Definition

Acquiring properties of parent class.

📝 Short Answer

Code reuse mechanism.

💻 Example

```java
class Animal {
    void eat() {
        System.out.println("Eating");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking");
    }
}
```

⭐ Why / Where / When Used

- Reusability
- Extending functionality

🔹 10. What is Polymorphism?

📘 Definition

Same method behaves differently.

📝 Short Answer

Many forms.

💻 Example

```java
class Animal {
    void sound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}
```

⭐ Why / Where / When Used

- Runtime flexibility
- Method overriding

🔹 11. What is Method Overloading?

📘 Definition

Same method name with different parameters.

📝 Short Answer

Compile-time polymorphism.

💻 Example

```java
class Calc {
    int add(int a, int b) { return a+b; }
    int add(int a, int b, int c) { return a+b+c; }
}
```

⭐ Why / Where / When Used

- Improves readability

🔹 12. What is Method Overriding?

📘 Definition

Subclass modifies parent method.

📝 Short Answer

Runtime polymorphism.

💻 Example

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

```
class Child extends Parent {
    void show() {
        System.out.println("Child");
    }
}
```

⭐ Why / Where / When Used

- Modify parent behavior

🔹 13. What is Constructor?

📘 Definition

Special method used to initialize object.

📝 Short Answer

Runs when object is created.

💻 Example

Java

```
class Student {
    Student() {
        System.out.println("Constructor called");
    }
}
```

⭐ Why / Where / When Used

- Object initialization

🔹 14. What is this Keyword?

📘 Definition

Refers to current object.

📝 Short Answer

Current object reference.

💻 Example

```java
class Student {
    int id;

    Student(int id) {
        this.id = id;
    }
}
```

⭐ Why / Where / When Used

- Avoid variable confusion

🔹 15. What is super Keyword?

📘 Definition

Refers to parent class object.

📝 Short Answer

Calls parent class members.

💻 Example

```java
class Parent {
    Parent() {
        System.out.println("Parent Constructor");
    }
}

class Child extends Parent {
    Child() {
        super();
    }
}
```

⭐ Why / Where / When Used

- Access parent methods

🔹 16. What is Static Keyword?

📘 Definition

Belongs to class rather than object.

📝 Short Answer

Shared memory.

💻 Example

```java
class Test {
    static int count = 0;
}
```

⭐ Why / Where / When Used

- Memory optimization

🔹 17. What is Final Keyword?

📘 Definition

Prevents modification.

📝 Short Answer

Constant or restricted inheritance.

💻 Example

```java
final class Test { }
```

⭐ Why / Where / When Used

- Security
- Immutable values

🔹 18. What is Exception Handling?

📘 Definition

Mechanism to handle runtime errors.

📝 Short Answer

Error control.

💻 Example

```java
try {
```

```
    int a = 10/0;
} catch(Exception e) {
    System.out.println("Error handled");
}
```

⭐ Why / Where / When Used

- Prevent program crash

🔹 19. Checked vs Unchecked Exception

📘 Definition

Checked → Compile time
Unchecked → Runtime

💻 Example

```java
// Checked
FileReader f = new FileReader("abc.txt");

// Unchecked
int a = 10/0;
```

⭐ Usage

- Checked → External resources
- Unchecked → Logic errors

🔹 20. What is Multithreading?

📘 Definition

Running multiple tasks simultaneously.

💻 Example

```java
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread running");
    }
}
```

⭐ Usage

- Performance improvement

🔹 21. What is Runnable Interface?

💻 Example

```java
class MyTask implements Runnable {
    public void run() {
        System.out.println("Task running");
    }
}
```

⭐ Usage

- Thread creation alternative

🔹 22. What is String Pool?

📘 Definition

Memory optimization technique.

💻 Example

```java
String s1 = "Java";
String s2 = "Java";
System.out.println(s1 == s2);
```

🔹 23. Difference Between String, StringBuilder, StringBuffer

💻 Example

```java
StringBuilder sb = new StringBuilder("Java");
sb.append(" World");
```

⭐ Usage

- String → Immutable
- StringBuffer → Thread safe
- StringBuilder → Fast

🔹 24. What is Collection Framework?

📘 Definition

Set of classes and interfaces for storing data.

💻 Example

```java
List<String> list = new ArrayList<>();
```

🔹 25. Difference Between List and Set

⭐ List

- Allows duplicates

⭐ Set

- Unique elements

🔹 26. Difference Between ArrayList and LinkedList

⭐ ArrayList

- Fast retrieval

⭐ LinkedList

- Fast insertion

🔹 27. What is Comparable?

💻 Example

```java
class Student implements Comparable<Student> {
    int id;

    public int compareTo(Student s) {
        return this.id - s.id;
    }
}
```

🔹 28. What is Comparator?

💻 Example

```java
Collections.sort(list, (a,b) -> a.id - b.id);
```

🔹 29. What is Stream API?

💻 Example

```java
list.stream()
    .filter(x -> x > 5)
    .forEach(System.out::println);
```

🔹 30. What is Lambda Expression?

📘 Definition

Short way to write anonymous function.

💻 Example

```java
Runnable r = () -> System.out.println("Lambda Running");
```

⭐ Usage

- Functional programming
- Stream operations
