**JAVA SCENARIO BASED QUESTIONS & ANSWERS**

**1. Scenario: Handling NullPointerException**

**Question:** You are given a method that processes user input, but sometimes it throws a NullPointerException. How would you handle this exception effectively?

**Answer:**

* First, I would check for null before using the object.
* I could use Optional to avoid direct null checks.
* If necessary, I would handle the exception using try-catch.

java

public void processUserInput(String input) {

if (input != null) {

System.out.println(input.toUpperCase());

} else {

System.out.println("Invalid input: null value provided");

}

}

Or using Optional:

java

import java.util.Optional;

public void processUserInput(String input) {

Optional.ofNullable(input)

.ifPresentOrElse(

val -> System.out.println(val.toUpperCase()),

() -> System.out.println("Invalid input: null value provided")

);

}

**2. Scenario: Deadlock Prevention**

**Question:** You are working on a multithreaded application where two threads are causing a deadlock. How would you prevent it?

**Answer:**

* Avoid nested locks and always acquire them in a fixed order.
* Use a timeout while acquiring locks.
* Use ReentrantLock instead of synchronized and try to get the lock without blocking indefinitely.

Example of avoiding deadlock:

java

class Resource {

private final Object lock1 = new Object();

private final Object lock2 = new Object();

public void method1() {

synchronized (lock1) {

synchronized (lock2) {

System.out.println("Executing method1");

}

}

}

public void method2() {

synchronized (lock1) { // Acquiring locks in the same order

synchronized (lock2) {

System.out.println("Executing method2");

}

}

}

}

**3. Scenario: Singleton Design Pattern**

**Question:** How would you implement a thread-safe Singleton class?

**Answer:**  
Using **double-checked locking**:

java

class Singleton {

private static volatile Singleton instance;

private Singleton() {} // Private constructor

public static Singleton getInstance() {

if (instance == null) {

synchronized (Singleton.class) {

if (instance == null) {

instance = new Singleton();

}

}

}

return instance;

}

}

**4. Scenario: Performance Optimization in a Large Collection**

**Question:** You have a large list of objects, and you need to find a specific object quickly. How would you optimize this?

**Answer:**

* Instead of a List, use a HashMap for O(1) lookup time.
* Use streams for parallel processing if applicable.

Example:

java

Map<Integer, Employee> employeeMap = employees.stream()

.collect(Collectors.toMap(Employee::getId, Function.identity()));

// Fast lookup

Employee emp = employeeMap.get(101);

**5. Scenario: Memory Leak Prevention**

**Question:** Your Java application has a memory leak. How would you find and fix it?

**Answer:**

* Use **profiling tools** like VisualVM or JProfiler.
* Avoid **unnecessary object references** (e.g., static collections).
* Use **WeakReferences** when appropriate.

Example of fixing memory leaks:

java

Map<String, WeakReference<Data>> cache = new HashMap<>();

cache.put("key1", new WeakReference<>(new Data()));

**6. Scenario: Immutable Class Implementation**

**Question:** How do you create an immutable class in Java?

**Answer:**

* Declare the class final.
* Make fields private final.
* Provide only getters and no setters.
* Do not return mutable objects from methods.

Example:

java

final class ImmutablePerson {

private final String name;

private final int age;

public ImmutablePerson(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() { return name; }

public int getAge() { return age; }

}

**7. Scenario: Processing Large Files**

**Question:** You need to read and process a large file without loading it entirely into memory. What would you do?

**Answer:**  
Use **BufferedReader** or **Java 8 Streams** to read line by line.

java

try (BufferedReader reader = new BufferedReader(new FileReader("largeFile.txt"))) {

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

Or using Java 8 Streams:

java

try (Stream<String> lines = Files.lines(Paths.get("largeFile.txt"))) {

lines.forEach(System.out::println);

} catch (IOException e) {

e.printStackTrace();

}

**8. Scenario: Concurrent Modification Exception**

**Question:** How would you handle ConcurrentModificationException while iterating a list?

**Answer:**

* Use Iterator's remove() method.
* Use CopyOnWriteArrayList for concurrent access.
* Use ConcurrentHashMap if modifying a map.

Example with Iterator:

java

List<String> list = new ArrayList<>(Arrays.asList("A", "B", "C"));

Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {

String item = iterator.next();

if (item.equals("B")) {

iterator.remove();

}

}

**9. Scenario: Producer-Consumer Problem**

**Question:** How would you implement a thread-safe producer-consumer pattern?

**Answer:**  
Use **BlockingQueue**:

java

import java.util.concurrent.\*;

class ProducerConsumer {

private static final BlockingQueue<Integer> queue = new LinkedBlockingQueue<>(5);

static class Producer implements Runnable {

public void run() {

try {

for (int i = 0; i < 5; i++) {

queue.put(i);

System.out.println("Produced: " + i);

Thread.sleep(1000);

}

} catch (InterruptedException e) { e.printStackTrace(); }

}

}

static class Consumer implements Runnable {

public void run() {

try {

while (true) {

int item = queue.take();

System.out.println("Consumed: " + item);

Thread.sleep(2000);

}

} catch (InterruptedException e) { e.printStackTrace(); }

}

}

public static void main(String[] args) {

new Thread(new Producer()).start();

new Thread(new Consumer()).start();

}

}

**10. Scenario: Preventing SQL Injection in Java**

**Question:** How would you prevent SQL injection in Java?

**Answer:**  
Use **PreparedStatement** instead of string concatenation:

java

String query = "SELECT \* FROM users WHERE username = ? AND password = ?";

try (Connection conn = DriverManager.getConnection(url, user, pass);

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, username);

stmt.setString(2, password);

ResultSet rs = stmt.executeQuery();

// Process result set

} catch (SQLException e) {

e.printStackTrace();

}

**11. Scenario: Thread Safety in Multi-Threaded Application**

**Q:** You are working on a banking application where multiple users update the account balance simultaneously. How will you ensure thread safety?

**A:**

* I would use **synchronized methods or blocks** to ensure that only one thread can update the balance at a time.
* Alternatively, I can use **Atomic variables (e.g., AtomicInteger, AtomicLong)** or **ReentrantLocks** for better performance.

**Example Using synchronized:**

java

public class BankAccount {

private double balance;

public synchronized void deposit(double amount) {

balance += amount;

}

public synchronized void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

} else {

throw new IllegalArgumentException("Insufficient funds");

}

}

}

**Example Using AtomicInteger:**

java

import java.util.concurrent.atomic.AtomicInteger;

public class BankAccount {

private AtomicInteger balance = new AtomicInteger(0);

public void deposit(int amount) {

balance.addAndGet(amount);

}

public void withdraw(int amount) {

balance.addAndGet(-amount);

}

}

**12. Scenario: Avoiding Memory Leaks in Java**

**Q:** Your application is experiencing high memory usage. How would you detect and fix memory leaks?

**A:**

* **Use tools like JVisualVM, JConsole, and Eclipse MAT** to analyze memory usage.
* **Avoid static references to large objects**, as they are not garbage-collected.
* **Close resources properly**, like database connections, file streams, etc.
* **Use WeakReferences** for objects that should be garbage-collected when no longer needed.

**Example: Closing Resources Properly (Try-With-Resources in Java 7+)**

java

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

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

**13. Scenario: Design a Singleton Class**

**Q:** How do you ensure a class is a singleton and thread-safe?

**A:**

* **Use an enum**, which is the best way to implement a Singleton in Java.
* **Use synchronized in a lazy initialization method.**
* **Use volatile to prevent multiple thread access issues.**

**Best Singleton Implementation (Using Enum):**

java

public enum Singleton {

INSTANCE;

public void showMessage() {

System.out.println("Singleton Instance");

}

}

Usage:

java

Singleton.INSTANCE.showMessage();

**14. Scenario: Optimizing Performance for Large Data Processing**

**Q:** You have to process a large list of data. How would you optimize performance?

**A:**

* **Use Java 8 Streams and Parallel Processing** to take advantage of multi-core CPUs.
* **Use batching instead of processing all data at once.**
* **Use caching techniques (e.g., LRU Cache, Guava Cache, Redis) to store frequently accessed data.**

**Example Using Parallel Streams:**

java

List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David");

names.parallelStream()

.map(String::toUpperCase)

.forEach(System.out::println);

**15. Scenario: Handling Large JSON Data Efficiently**

**Q:** You need to parse a huge JSON file (e.g., 1GB). How do you handle this efficiently?

**A:**

* **Use Jackson Streaming API instead of ObjectMapper to avoid memory issues.**
* **Read the file in chunks instead of loading it all into memory.**

**Example Using Jackson Streaming API:**

java

import com.fasterxml.jackson.core.JsonFactory;

import com.fasterxml.jackson.core.JsonParser;

import com.fasterxml.jackson.core.JsonToken;

import java.io.File;

import java.io.IOException;

public class LargeJsonReader {

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

JsonFactory factory = new JsonFactory();

JsonParser parser = factory.createParser(new File("largeFile.json"));

while (parser.nextToken() != JsonToken.END\_OBJECT) {

String fieldName = parser.getCurrentName();

if ("name".equals(fieldName)) {

parser.nextToken();

System.out.println(parser.getText());

}

}

parser.close();

}

}