EXCEPTION HANDLING — CRISP NOTES

□ Why Exception Handling?

To stop program crash → handle abnormal conditions → run smoothly.

□ Types of Exceptions

✔ Checked Exceptions

- Compile–time checked
- You MUST handle using try/catch OR throws
- Example: IOException, SQLException

✔ Unchecked Exceptions

- Runtime errors
- No compile-time checking
- Example: NullPointerException, ArithmeticException

✔ Errors

- Irrecoverable → cannot handle
- Example: StackOverflowError, OutOfMemoryError

□ Exception Keywords

try

→ Code which may throw exception

catch

→ Handle exception

finally

→ Always executes (DB close, file close)

throw

→ Throw exception manually

throws

→ Pass exception responsibility to caller

□ Important Interview Points

- Checked → Compile-time
- Unchecked → Runtime
- finally always runs (except System.exit)
- throw = create exception
- throws = declare exception
- Custom exceptions extend **Exception** / **RuntimeException**
- Never catch Exception blindly unless necessary

□ MULTITHREADING — CRISP NOTES

□ Why Multithreading?

To do tasks in parallel → better performance → responsive applications.

□ How to Create Thread?

1□ extends Thread

```
class MyThread extends Thread { public void run(){} }
```

2□ implements Runnable

```
new Thread(new MyRunnable()).start();
```

3□ Using Lambda

```
new Thread(() -> {}).start();
```

□ Thread Lifecycle

```
New → Runnable → Running → Blocked/Waiting → Terminated
```

□ Important Thread Methods

- **start()** → starts new thread
- **run()** → thread logic
- **sleep(ms)** → pause
- **join()** → wait for other thread
- **interrupt()** → ask thread to stop

□ Race Condition & synchronized

When 2 threads access shared data → inconsistent results → **race condition**.

Solution:

```
synchronized void method() {}
```

Ensures only ONE thread enters the block.

□ volatile Keyword

Used for **visibility** between threads.
Guarantees latest value read by all threads.

Example:

```
volatile boolean flag = true;
```

□ wait(), notify(), notifyAll()

Used for **thread communication** inside synchronized block.

- wait() → release lock + wait
- notify() → wake one thread
- notifyAll() → wake all threads

□ Deadlock

Two threads waiting on each other's lock → infinite waiting.

Fix:
✔ Lock ordering
✔ Avoid nested locks

□ ExecutorService (ThreadPool)

Better alternative to manually creating threads.

```
ExecutorService pool = Executors.newFixedThreadPool(5);
```

Benefits:
✔ Manages threads
✔ Reuses threads
✔ Prevents performance issues

□ FINAL INTERVIEW REVISION (SUPER SHORT)

Exceptions

- Checked = compile-time
- Unchecked = runtime
- Errors = unhandleable
- try/catch/finally
- throw vs throws
- finally always runs

Multithreading

- Thread creation: Thread, Runnable, Lambda
- synchronized = thread safety
- volatile = visibility
- wait/notify = communication
- Deadlock = circular waiting
- ThreadPool = best practice