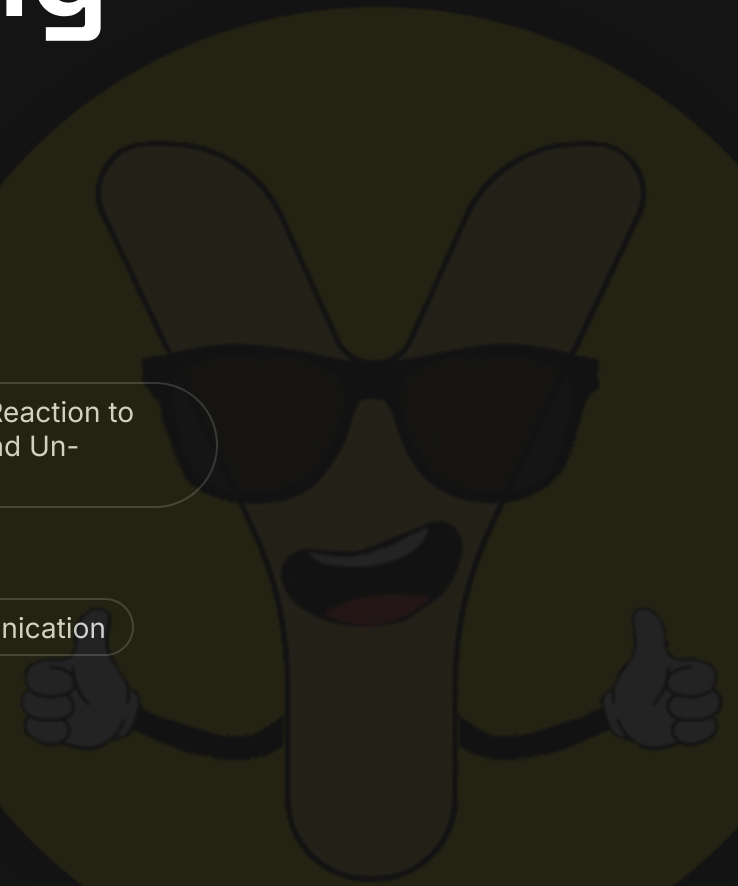
AKTU · 2ND YEAR · SEMESTER 4 · BCS403

Object Oriented Programming with Java

Unit 2 — Exception Handling, Input/Output Basics, and Multithreading

- Exception Handling: The Idea behind Exception, Exceptions & Errors, Types of Exception, Control Flow in Exceptions, JVM Reaction to Exceptions, Use of try, catch, finally, throw, throws in Exception Handling, In-built and User Defined Exceptions, Checked and Un-Checked Exceptions
- Input / Output Basics: Byte Streams and Character Streams, Reading and Writing File in Java
- Multithreading: Thread, Thread Life Cycle, Creating Threads, Thread Priorities, Synchronizing Threads, Inter-thread Communication



DECODED BY

Chirag Jain

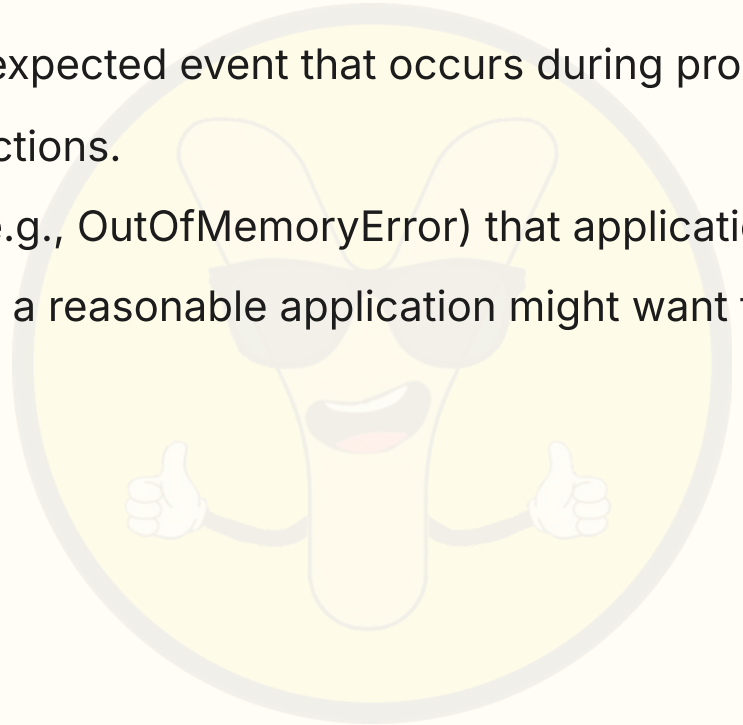
Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

theycotes.com

Exception Handling: Basics

Exception handling is a mechanism to handle runtime errors to maintain normal program flow.

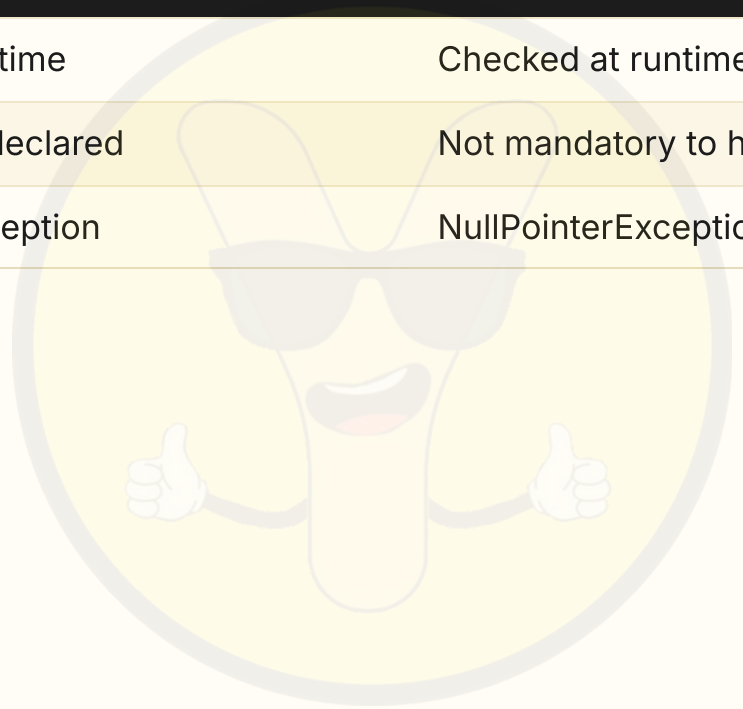
- An Exception is an unwanted or unexpected event that occurs during program execution.
- It disrupts the normal flow of instructions.
- Error: Indicates serious problems (e.g., OutOfMemoryError) that applications should not try to catch.
- Exception: Indicates conditions that a reasonable application might want to catch.



Exception Hierarchy

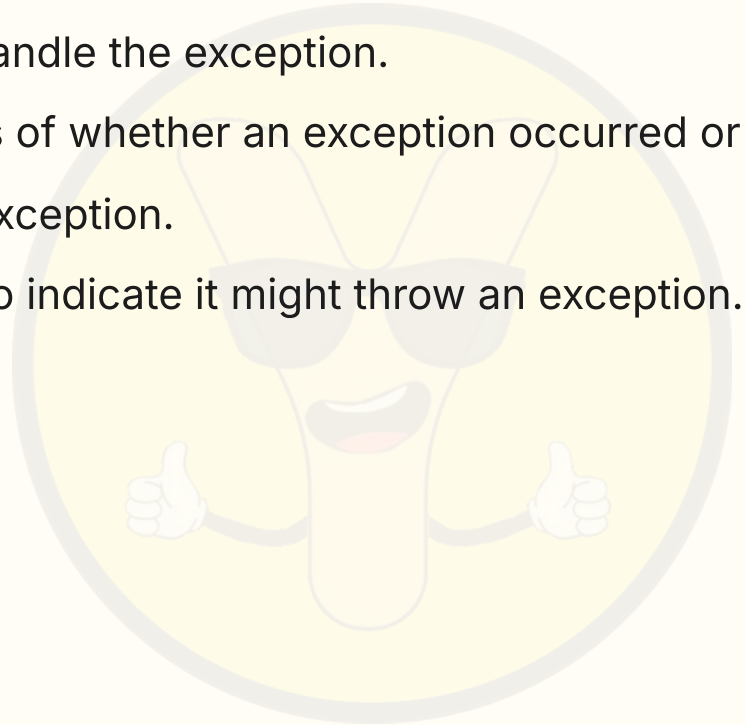
Checked vs Unchecked Exceptions

Feature	Checked Exception	Unchecked Exception
Definition	Checked at compile-time	Checked at runtime
Requirement	Must be handled or declared	Not mandatory to handle
Example	IOException, SQLException	NullPointerException, ArithmeticException



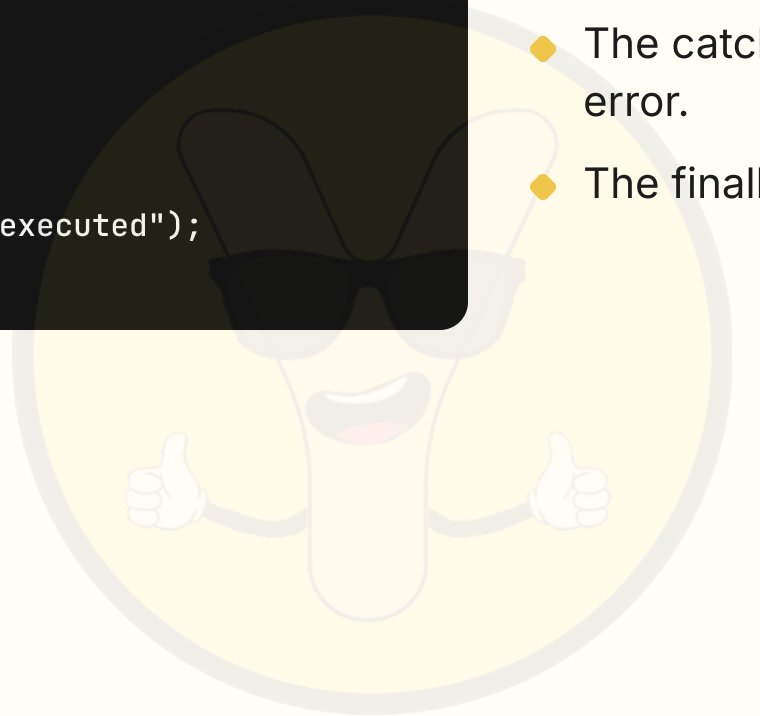
Exception Keywords

- try: Defines a block of code to be tested for errors.
- catch: Defines a block of code to handle the exception.
- finally: Always executes, regardless of whether an exception occurred or not.
- throw: Used to explicitly throw an exception.
- throws: Used in method signature to indicate it might throw an exception.



Exception Handling Example

```
try {
    int data = 100 / 0;
} catch (ArithmeticException e) {
    System.out.println(e);
} finally {
    System.out.println("Finally block executed");
}
```

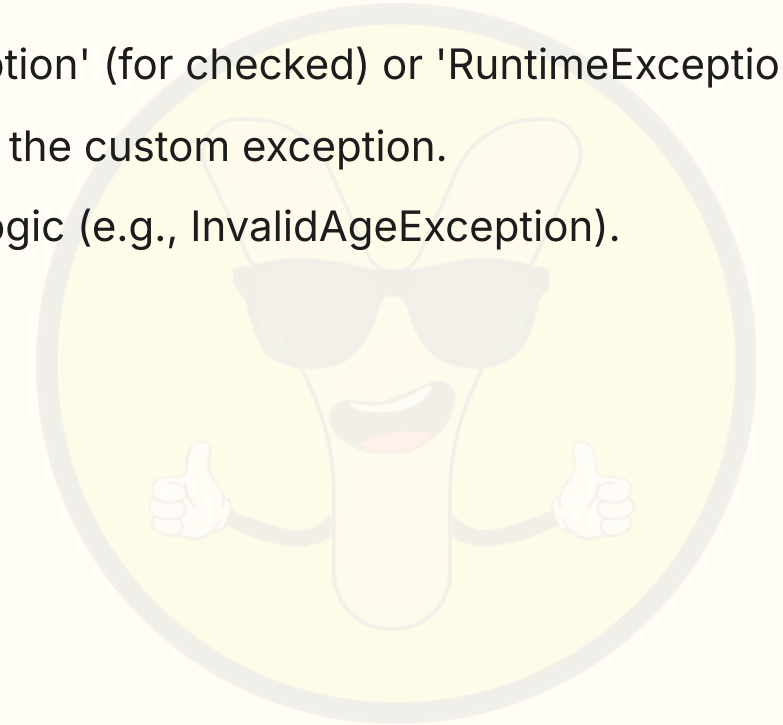


- The try block contains the risky code.
- The catch block handles the division by zero error.
- The finally block ensures resource cleanup.

User-Defined Exceptions

Java allows creating custom exceptions by extending the Exception class.

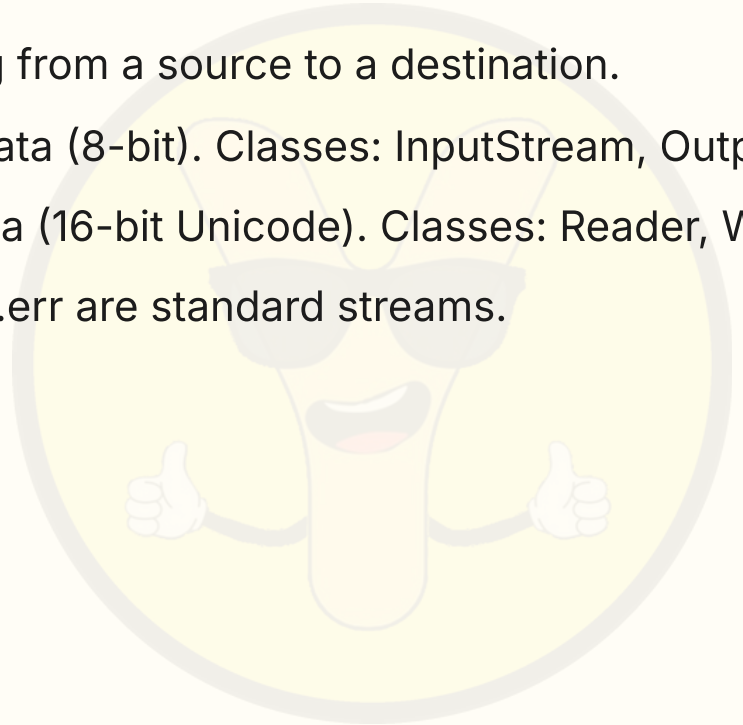
- Create a class that extends 'Exception' (for checked) or 'RuntimeException' (for unchecked).
- Use the 'throw' keyword to trigger the custom exception.
- Useful for domain-specific error logic (e.g., InvalidAgeException).



Input/Output Basics

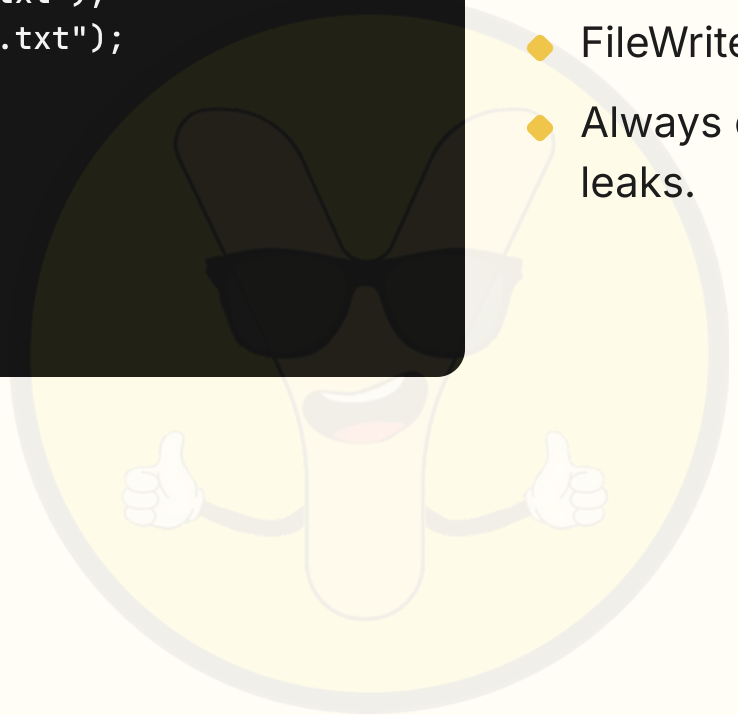
Java I/O handles data transfer using Streams.

- Stream: A sequence of data flowing from a source to a destination.
- Byte Stream: Used for raw binary data (8-bit). Classes: InputStream, OutputStream.
- Character Stream: Used for text data (16-bit Unicode). Classes: Reader, Writer.
- System.in, System.out, and System.err are standard streams.



File Reading and Writing

```
FileReader fr = new FileReader("input.txt");
FileWriter fw = new FileWriter("output.txt");
int c;
while ((c = fr.read()) ≠ -1) {
    fw.write(c);
}
fr.close();
fw.close();
```

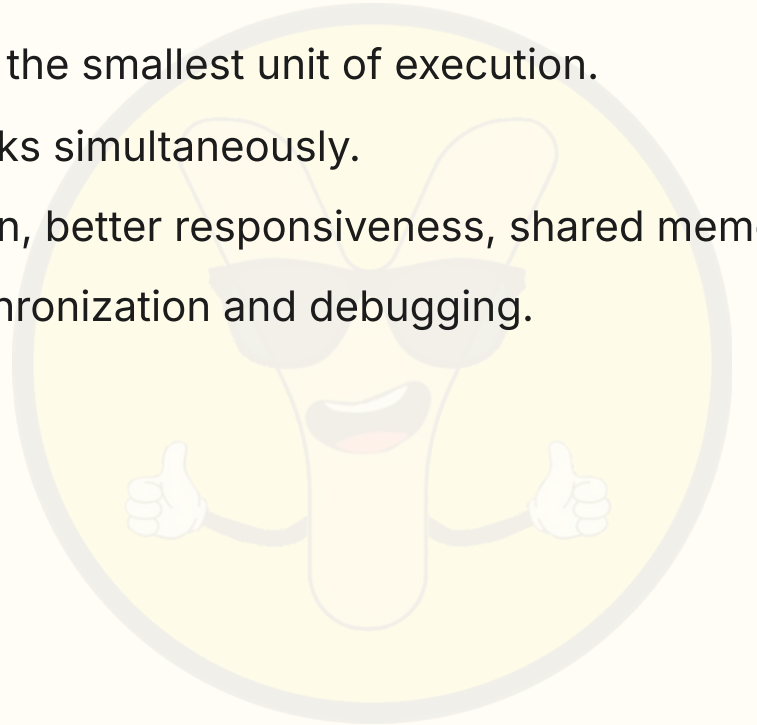


- FileReader reads character by character.
- FileWriter writes character by character.
- Always close streams to prevent memory leaks.

Multithreading: Introduction

Multithreading is the ability of a CPU to execute multiple threads concurrently.

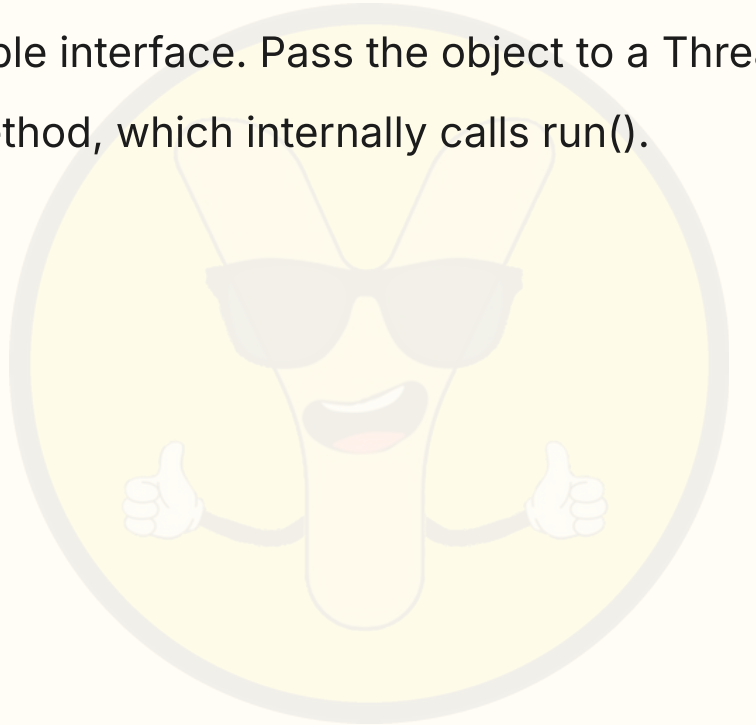
- Thread: A lightweight sub-process, the smallest unit of execution.
- Multitasking: Executing multiple tasks simultaneously.
- Advantages: Efficient CPU utilization, better responsiveness, shared memory resources.
- Disadvantages: Complexity in synchronization and debugging.



Thread Life Cycle

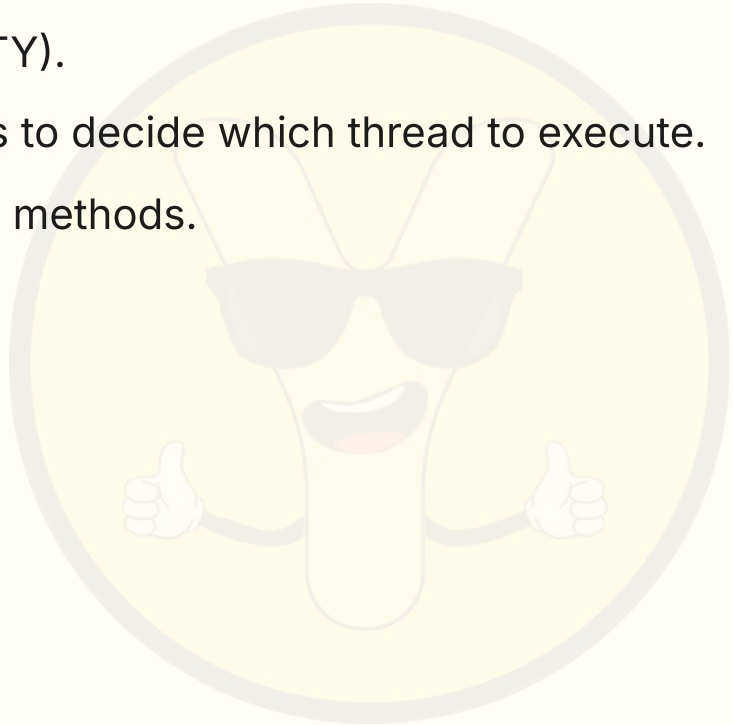
Creating Threads

- Method 1: Extending the Thread class. Override the run() method.
- Method 2: Implementing the Runnable interface. Pass the object to a Thread constructor.
- Starting a thread: Call the start() method, which internally calls run().



Thread Priorities

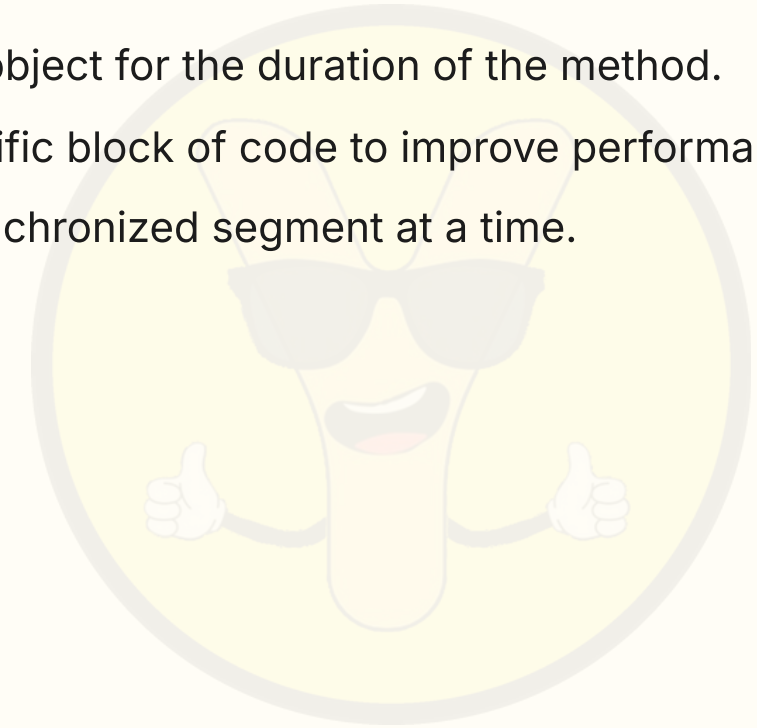
- Each thread has a priority ranging from 1 (MIN_PRIORITY) to 10 (MAX_PRIORITY).
- Default priority is 5 (NORM_PRIORITY).
- The thread scheduler uses priorities to decide which thread to execute.
- Use setPriority(int) and getPriority() methods.



Synchronization

Synchronization prevents thread interference and memory consistency errors.

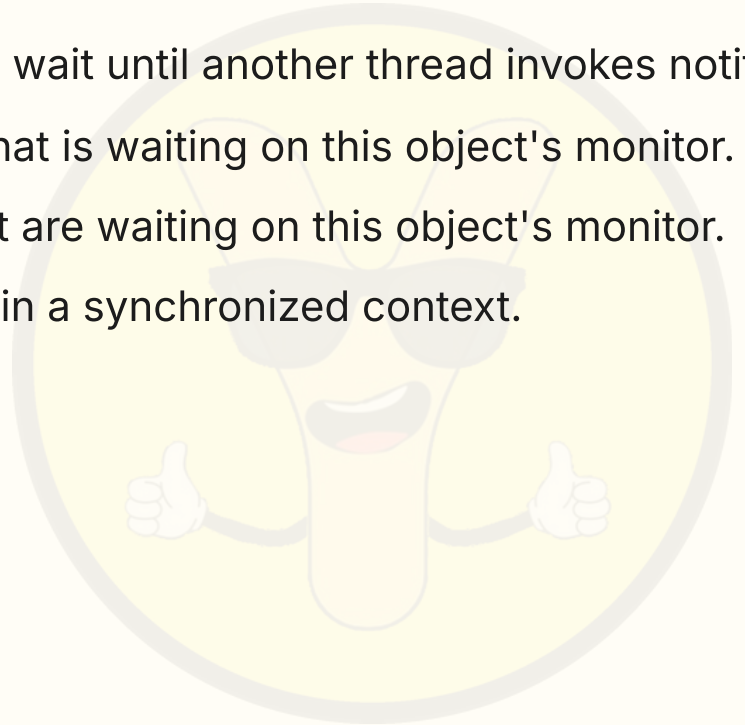
- Synchronized Method: Locks the object for the duration of the method.
- Synchronized Block: Locks a specific block of code to improve performance.
- Only one thread can execute a synchronized segment at a time.



Inter-thread Communication

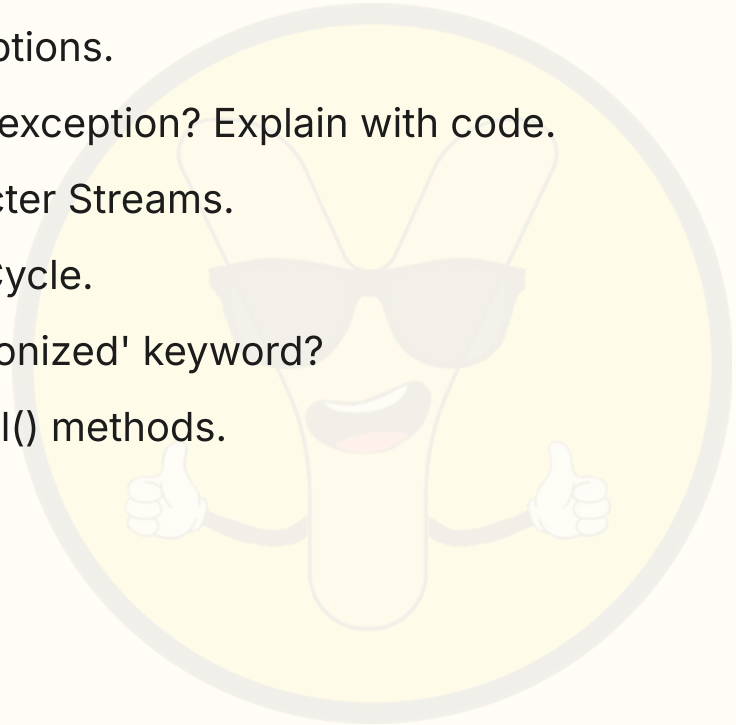
Mechanism for threads to talk to each other.

- wait(): Causes the current thread to wait until another thread invokes notify().
- notify(): Wakes up a single thread that is waiting on this object's monitor.
- notifyAll(): Wakes up all threads that are waiting on this object's monitor.
- These methods must be called within a synchronized context.



Exam corner

- **Q1** Differentiate between Error and Exception with examples.
- **Q2** Explain the hierarchy of Java exceptions.
- **Q3** How do you create a user-defined exception? Explain with code.
- **Q4** Compare Byte Streams and Character Streams.
- **Q5** Draw and explain the Thread Life Cycle.
- **Q6** What is the purpose of the 'synchronized' keyword?
- **Q7** Explain wait(), notify(), and notifyAll() methods.



Unit 2 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 3 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

