1. **What does the expression “try … catch” do?**

The try ... catch block in Java is used for **exception handling**. It helps prevent program crashes by catching and handling runtime errors (exceptions) that might occur in the try block.

**How it works:**

* **try block**: Contains the code that might throw an exception.
* **catch block**: Specifies how to handle a specific type of exception.

**Purpose:**

* Ensures the program does not terminate unexpectedly.
* Provides an opportunity to recover from errors or notify users.

**Example:**

*public class Main {*

*public static void main(String[] args) {*

*try {*

*int result = 10 / 0; // This will throw an ArithmeticException*

*System.out.println("Result: " + result);*

*} catch (ArithmeticException e) {*

*System.out.println("Error: Cannot divide by zero.");*

*}*

*}*

*}*

**Output:**

*Error: Cannot divide by zero.*

1. **What does it mean when the program throws exceptions?**

When a program **throws an exception**, it means an **error has occurred** during execution, and the program cannot proceed normally. Exceptions are Java's way of signaling that something went wrong.

**Types of Exceptions:**

1. **Checked Exceptions**:
   * Must be declared in the method signature or handled with try-catch.
   * Example: IOException, SQLException.
2. **Unchecked Exceptions**:
   * Occur at runtime and do not need to be declared or explicitly handled.
   * Example: NullPointerException, ArithmeticException.
3. **Errors**:
   * Represent serious problems that applications usually should not try to handle.
   * Example: OutOfMemoryError.

**What happens when an exception is thrown?**

1. Java stops the normal flow of the program.
2. It looks for a matching catch block to handle the exception.
3. If no handler is found, the program terminates and displays the exception stack trace.

**Example:**

*public class Main {*

*public static void main(String[] args) {*

*int[] numbers = {1, 2, 3};*

*try {*

*System.out.println(numbers[5]);*

*// Throws ArrayIndexOutOfBoundsException*

*} catch (ArrayIndexOutOfBoundsException e) {*

*System.out.println("Error: Array index is out of bounds.");*

*}*

*}*

*}*

**Output:**

*Error: Array index is out of bounds.*

1. **What is a Thread in Java?**

A **Thread** in Java is a lightweight unit of a process. Threads allow a program to perform multiple tasks **concurrently** (at the same time).

**Key Concepts:**

1. **Multithreading**: The ability to run multiple threads simultaneously within a single program.
2. **Thread Lifecycle**: A thread can be in one of the following states:
   * **New**: Created but not yet started.
   * **Runnable**: Ready to run but waiting for CPU time.
   * **Running**: Actively executing.
   * **Blocked/Waiting**: Waiting for a resource or signal.
   * **Terminated**: Completed execution.

**Why use Threads?**

* Improves program efficiency by utilizing CPU more effectively.
* Useful for tasks like file reading, network operations, and UI interactions.

**How to create a Thread:**

1. **Extend the Thread class**:

*class MyThread extends Thread {*

*@Override*

*public void run() {*

*System.out.println("Thread is running.");*

*}*

*}*

*public class Main {*

*public static void main(String[] args) {*

*MyThread thread = new MyThread();*

*thread.start(); // Start the thread*

*}*

*}*

1. **Implement the Runnable interface**:

class MyRunnable implements Runnable {

@Override

public void run() {

System.out.println("Thread is running.");

}

}

public class Main {

public static void main(String[] args) {

Thread thread = new Thread(new MyRunnable());

thread.start(); // Start the thread

}

}

**Output for both examples:**

*Thread is running.*

**Common Uses:**

* Parallel processing
* Background tasks
* Responsive user interfaces (e.g., GUI applications)