Overview: Errors, Exceptions, Threads & File Handling

Presented By:
Mirazul Islam
Java Instructor, EU

Types of Errors in Programming

Major types:

- Syntax Error
- Logical Error
- System Error

Syntax Error

Definition:

An error that occurs when code violates the rules of the programming language.

Detected:

At compile time (or before execution in interpreted languages).

Characteristics:

- Program won’t compile or run.
- Easy to detect and fix.

```
int x = 10  // Missing semicolon
Console.WriteLine(x)  // Missing semicolon
if (x > 10  // Missing closing parenthesis
{
    // code
}
```

Logical Error

Definition:

Code runs without crashing but produces incorrect results due to flawed logic.

Detected:

At runtime, through unexpected behavior/output.

Characteristics:

- Harder to identify.
- No error message shown.

```
int x = 5;

int square = x + x;  // Logical Error: Should be x * x
```

System Error (Fatal Error)

Definition:

Serious problems not handled by the program, often due to hardware or environment issues.

Characteristics:

- Not recoverable.
- Often causes program or system crash.

Examples:

- StackOverflowError
- OutOfMemoryError
- System crash due to corrupted memory

What is an Exception?

Definition: Conditions that disrupt program flow but can be handled.

Examples:

- File not found
- Divide by zero
- Invalid user input

Characteristics:

- Often predictable.
- Can be caught and handled gracefully.

Exception Handling Mechanism

- **try** – code block where exception may occur.
- **catch** – handles the exception.
- **finally** – optional block that executes regardless of exception.
- **throw** – used to explicitly raise an exception.

Example Code – Try-Catch in Java

```java
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            int a = 10;
            int b = 0;
            int result = a / b;  // This will throw ArithmeticException
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero.");
        } finally {
            System.out.println("Operation completed.");
        }
    }
}
```

What is a Thread?

- A thread is a lightweight subprocess.
- It is the smallest unit of execution.
- Threads run concurrently within a program.
- Java supports multithreading natively.

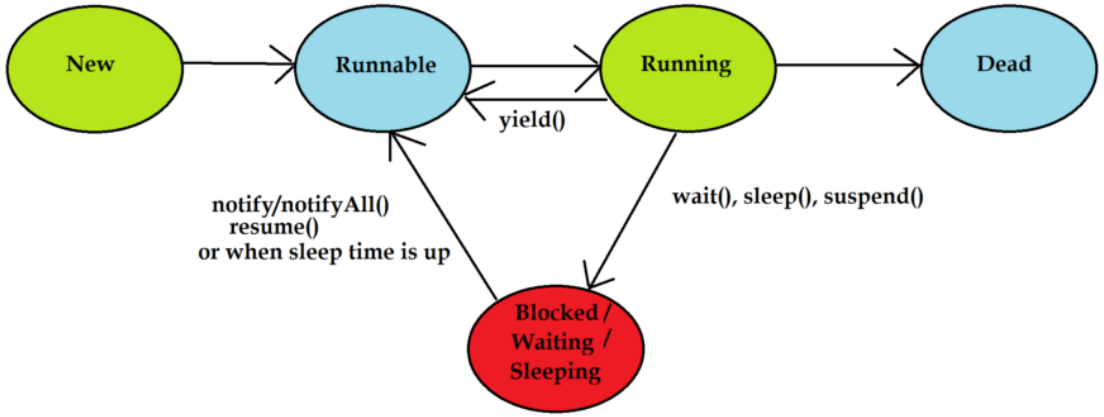
Benefits of Multithreading

- Improved application performance.
- Efficient use of CPU resources.
- Better user experience with responsive UI.
- Useful for performing multiple tasks simultaneously.

Java Thread Lifecycle

States:

- **New** – Thread is created**.**
- **Runnable** – Ready to run.
- **Running** – Currently executing.
- **Blocked/Waiting** – Waiting for resources.
- **Terminated** – Execution finished.



Thread Lifecycle using Thread states

Creating Threads in Java

- **Extending Thread class**

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

- **Implementing Runnable interface**

```
class MyRunnable implements Runnable {
  public void run() {
    System.out.println("Thread running");
  }
}
```

Starting a Thread

```
MyThread t1 = new MyThread();

t1.start(); // Starts a new thread


// OR

Thread t2 = new Thread(new MyRunnable());

t2.start();
```

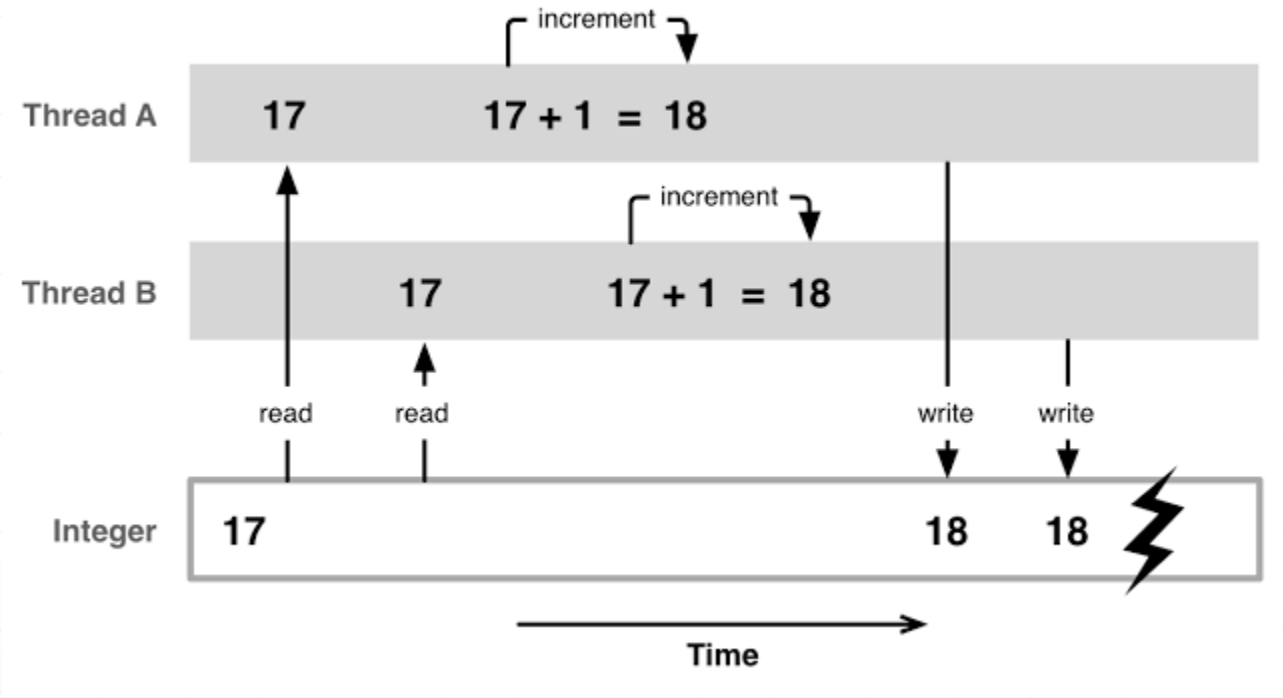
Thread Methods

- start() – Starts the thread.
- run() – Defines thread's code.
- sleep(ms) – Pauses thread for milliseconds.
- join() – Waits for thread to die.
- isAlive() – Checks if thread is alive.

Synchronization

- Prevents race conditions.
- Ensures thread safety.

```
synchronized void myMethod() {
  // thread-safe code
}
```



Thread Priorities

- Values: MIN_PRIORITY, NORM_PRIORITY, MAX_PRIORITY

Constant	Value	Description
Thread.MIN_PRIORITY	1	Lowest priority
Thread.NORM_PRIORITY	5	Default priority for threads
Thread.MAX_PRIORITY	10	Highest priority

```
public class PriorityExample {
    public static void main(String[] args) {
        Thread high = new Thread(() -> System.out.println("High priority thread"));
        Thread low = new Thread(() -> System.out.println("Low priority thread"));

        high.setPriority(Thread.MAX_PRIORITY); // 10
        low.setPriority(Thread.MIN_PRIORITY);  // 1

        high.start();
        low.start();
    }
}
```

```java
class ThreadTest extends Thread{  4 usages
    private String ThreadName;  2 usages
    ThreadTest(String name){  2 usages
        ThreadName = name;
    }
    @Override
    public void run(){
        for(int i = 1; i <= 5; i ++){
            System.out.println(ThreadName +" " + i);
            try{
                Thread.sleep( millis: 1000);
            }
            catch (Exception ex){
                System.out.println("Interept");
            }
        }
    }
}
public class Main {
    public static void main(String[] args) {
        ThreadTest t1 = new ThreadTest( name: "Thread A");
        ThreadTest t2 = new ThreadTest( name: "Thread B");

        t1.start();
        t2.start();
    }
```

```java
class ThreadTest implements Runnable{  4 usages
    private String ThreadName;  2 usages
    ThreadTest(String name){  2 usages
        ThreadName = name;
    }
    @Override
    public void run(){
        for(int i = 1; i <= 5; i ++){
            System.out.println(ThreadName +" " + i);
            try{
                Thread.sleep( millis: 1000);
            }
            catch (Exception ex){
                System.out.println("Interept");
            }
        }
    }
}
public class Main {
    public static void main(String[] args) {
        ThreadTest task1 = new ThreadTest( name: "Thread A");
        ThreadTest task2 = new ThreadTest( name: "Thread B");
        Thread t1 = new Thread(task1);
        Thread t2 = new Thread(task2);
        t1.start();
        t2.start();
```

Real-Life Examples

- Web servers handling multiple client requests.
- Video games with separate threads for rendering, input, and audio.
- Background data processing in mobile apps.

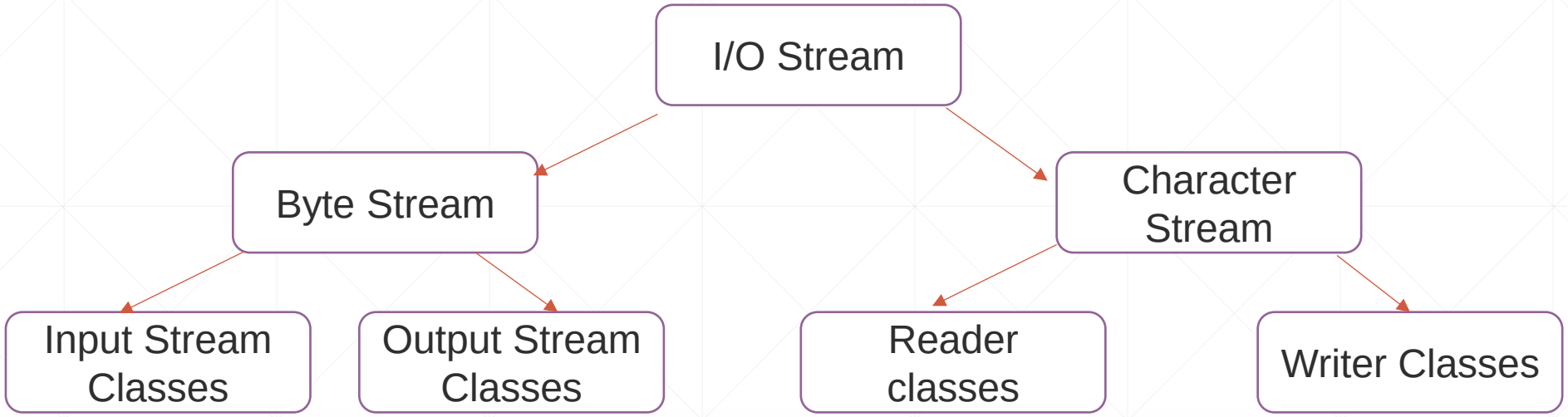
Introduction to File Handling

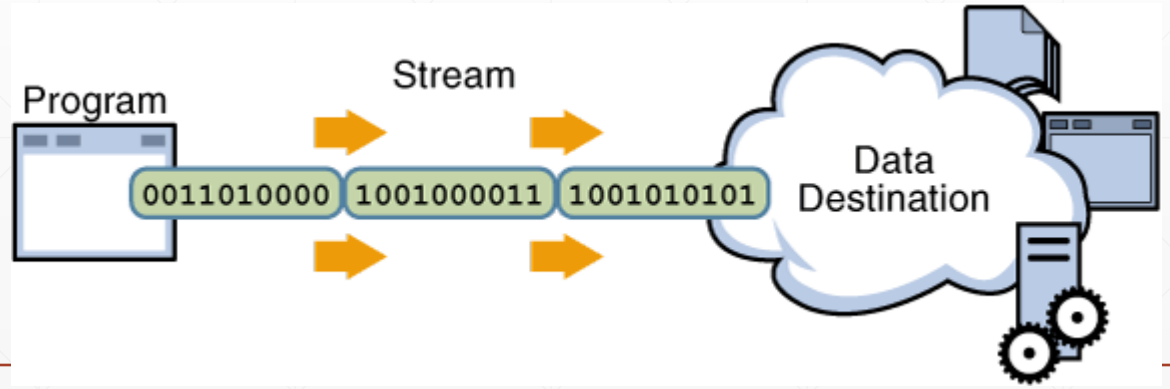
- Java provides built-in classes for file handling.
- Common file operations include reading, writing, and modifying files.
- File handling in Java can be done using:
  i. File class (for file properties and operations).
  ii. FileReader, FileWriter, BufferedReader, BufferedWriter.
  iii. NIO (New I/O) for advanced file handling.

What is a Stream?

- A stream is a sequence of data that flows from a source to a destination.
- Used in file handling for input and output operations.
- **Types of Streams:**
- Input Stream → Reads data (e.g., FileInputStream, BufferedReader)
- Output Stream → Writes data (e.g., FileOutputStream, BufferedWriter)

Streams


I/O Stream
Byte Stream
Character Stream
Input Stream Classes
Output Stream Classes
Reader classes
Writer Classes


Program
Stream
0011010000
1001000011
1001010101
Data Destination

Java Methods Use in File Type

- createNewFile() – Creates a new empty file.
- exists() – Checks if the file exists.
- delete() – Deletes a file.
- renameTo(File dest) – Renames the file.
- canRead(), canWrite() – Checks read/write permission.
- length() – Returns the file size in bytes.
- getName(), getPath() – Returns file name/path.
- isFile(), isDirectory() – Checks file or directory type.

Create a File

```
public class Main {
    public static void main(String[] args) {
    try {
        File myFile = new File( pathname: "example.txt");
        if (myFile.createNewFile())
            System.out.println("File created");
        else
            System.out.println("File already exist");
    }
    catch (Exception ex){
        System.out.println("exception");
    }

    }
}
```

Java File Writer

```java
public class Main {
    public static void main(String[] args) {
        try {
            FileWriter myWritter = new FileWriter( fileName: "example.txt");
            myWritter.write( str: "Hello World");
            myWritter.close();
            System.out.println("Successfully wrote in file");
        }
        catch (Exception ex){
            System.out.println("exception");
        }
    }
}
```

Java File Reader

```java
public class Main {
    public static void main(String[] args) {
        try {
            File obj = new File( pathname: "example.txt");
            Scanner scanner = new Scanner(obj);
            while(scanner.hasNextLine()){
                String data = scanner.nextLine();
                System.out.println(data);
            }
            scanner.close();
        }
        catch (Exception ex){
            System.out.println("exception");
        }
    }
}
```