Download Manager Simulator: Project Report

Course: Java Programming / Operating Systems / Multithreading in Java
Student Name: [Your Name]
Roll Number: [Your Roll Number]
Date of Submission: November 24, 2025
Institute: VIT (Vellore Institute of Technology)

Abstract

This project demonstrates the fundamentals of Java multithreading through a Download Manager Simulator. The application simulates concurrent file downloads using multiple threads, illustrating how the Runnable interface and Thread class enable parallel task execution. Three files are downloaded simultaneously, each on a separate thread, while the main thread remains free to perform other tasks. This project emphasizes core multithreading concepts including thread creation, thread management, and concurrent task execution without blocking the main application flow.

---

1. Introduction

1.1 Problem Statement

In traditional sequential programming, tasks are executed one after another. When downloading multiple files, the program must wait for one download to complete before starting the next. This is inefficient and wastes resources. Multithreading allows multiple downloads to happen simultaneously, improving application responsiveness and resource utilization[1][2].

1.2 Project Objective

The primary objective of this project is to:

- Understand and implement basic multithreading concepts in Java
- Demonstrate parallel execution of multiple file downloads
- Show how the main thread remains responsive while background tasks execute
- Provide hands-on experience with the Runnable interface and Thread class
- Illustrate the difference between sequential and concurrent execution

1.3 Scope

This project focuses on fundamental multithreading concepts without advanced features like thread pools, synchronization, or inter-thread communication. It is designed as an educational simulator, not a production-level download manager.

---

2. Multithreading Concepts

2.1 What is Multithreading?

Multithreading is a Java feature that allows a program to execute multiple threads simultaneously within the same process. Each thread is an independent path of execution that runs concurrently with other threads, sharing memory and system resources[1].

2.2 Key Concepts

Thread: A lightweight unit of execution representing a single path of code that runs independently.

Runnable Interface: An interface with a single `run()` method. Classes implementing `Runnable` define the task to be executed by a thread.

Thread Class: Java's built-in class used to create and manage threads. It can be instantiated by passing a `Runnable` object.

Concurrent Execution: Multiple threads running simultaneously, appearing to execute in parallel on multi-core systems or through time-slicing on single-core systems.

start() Method: Starts a new thread and calls its `run()` method asynchronously. Direct calls to `run()` execute in the current thread, not a new one.

Thread.sleep(): Pauses the current thread for a specified number of milliseconds, useful for simulating time-consuming operations[2].

2.3 Why Multithreading?

- **Responsiveness:** Application remains responsive while long operations run in background threads
- **Resource Sharing:** Threads share memory, reducing overhead compared to processes
- **Parallel Execution:** Multiple tasks execute truly in parallel on multi-core processors
- **Efficient Resource Utilization:** Better use of CPU time and system resources

---

3. Project Design

3.1 Architecture Overview

The project consists of two main classes:

FileDownloader Class

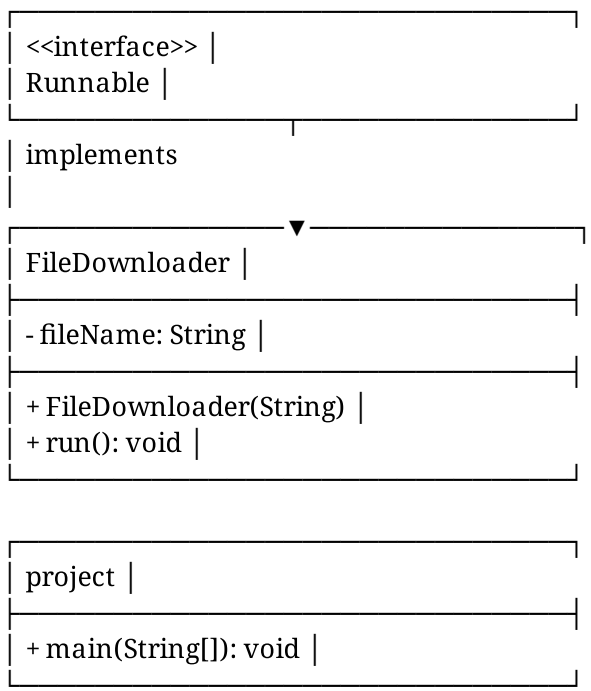
- Implements the `Runnable` interface
- Defines the download task for a single file
- Contains the logic for simulating file download progress

project Class

- Contains the main method
- Creates and starts multiple threads

- Demonstrates non-blocking behavior of the main thread

3.2 Class Diagram



(Uses Thread objects to start FileDownloader tasks)

3.3 Execution Flow

1. main() starts execution
2. Create three FileDownloader instances:
    - FileDownloader("File1.pdf")
    - FileDownloader("File2.mp4")
    - FileDownloader("File3.zip")
3. Wrap each in a Thread object with unique thread names
4. Call start() on each thread
5. Main thread prints "Main thread is free to do other tasks..."
6. All three download threads run concurrently
7. Each thread independently:
    - Prints starting message
    - Simulates 5 download steps (20%-100%)
    - Sleeps 500ms between steps
    - Prints completion message
8. Program terminates when all threads complete

---

4. Implementation Details

4.1 FileDownloader Class

Purpose: Implements the `Runnable` interface to define the download task.

Key Methods:

public FileDownloader(String fileName)

- Constructor: Initializes the FileDownloader with a file name

public void run()

- Called when the thread starts
- Simulates file download in 5 steps (20%, 40%, 60%, 80%, 100%)
- Each step includes a 500ms delay using Thread.sleep()
- Prints progress updates with thread name
- Handles InterruptedException if thread is interrupted

Key Features:

- Simple and focused: Only handles one file download
- Error handling: Catches `InterruptedException` for graceful interruption
- Thread-aware: Prints thread name using `Thread.currentThread().getName()`
- Progress simulation: Uses a loop with percentage updates

4.2 project Class

Purpose: Main class that creates and manages multiple download threads.

Key Method:

public static void main(String[] args)

- Prints startup message
- Creates three FileDownloader instances
- Wraps each in a Thread object with custom thread names
- Calls start() on all threads
- Prints message showing main thread independence

4.3 Code Walkthrough

Creating Threads:

```
Thread file1 = new Thread(new FileDownloader("File1.pdf"), "Thread-1");
Thread file2 = new Thread(new FileDownloader("File2.mp4"), "Thread-2");
Thread file3 = new Thread(new FileDownloader("File3.zip"), "Thread-3");
```

Each line creates a Thread object that:

- Takes a FileDownloader (Runnable) as the task
- Assigns a unique thread name for identification

Starting Threads:
file1.start();
file2.start();
file3.start();

Calling `start()` tells the JVM to create new threads and execute `run()` asynchronously.

Main Thread Continues:
System.out.println("Main thread is free to do other tasks...\n");

This executes immediately without waiting for download threads to finish, demonstrating non-blocking behavior.

---

5. Execution and Output

5.1 Sample Output

Download Manager Started

Starting download: File1.pdf on Thread-1
Starting download: File2.mp4 on Thread-2
Starting download: File3.zip on Thread-3
Main thread is free to do other tasks...

Downloading File1.pdf ... 20% completed
Downloading File2.mp4 ... 20% completed
Downloading File3.zip ... 20% completed
Downloading File1.pdf ... 40% completed
Downloading File2.mp4 ... 40% completed
Downloading File3.zip ... 40% completed
Downloading File1.pdf ... 60% completed
Downloading File2.mp4 ... 60% completed
Downloading File3.zip ... 60% completed
Downloading File1.pdf ... 80% completed
Downloading File2.mp4 ... 80% completed
Downloading File3.zip ... 80% completed
Downloading File1.pdf ... 100% completed
Downloading File2.mp4 ... 100% completed
Downloading File3.zip ... 100% completed
Download completed: File1.pdf on Thread-1
Download completed: File2.mp4 on Thread-2
Download completed: File3.zip on Thread-3

5.2 Output Analysis

Key Observations:

1. **Concurrent Execution:** All three downloads show progress updates interleaved, not sequential
2. **Thread Names:** Each line clearly shows which thread is executing (Thread-1, Thread-2, Thread-3)
3. **Main Thread Message:** Appears early, showing the main thread is not blocked

4. **Variable Order:** The exact sequence of output varies between runs due to thread scheduling by the JVM
5. **Identical Progress:** All files reach the same percentage at roughly the same time, confirming parallel execution

5.3 Compilation and Execution

Compile:
javac project.java

Run:
java project

---

6. Key Learning Outcomes

6.1 Concepts Demonstrated

✓ **Thread Creation:** Using the `Thread` class with `Runnable` interface
✓ **Concurrent Execution:** Multiple tasks running simultaneously
✓ **Non-Blocking Main Thread:** Main program remains responsive
✓ **Thread Management:** Naming and identifying threads
✓ **Simulation:** Using `Thread.sleep()` to simulate real operations
✓ **Exception Handling:** Proper handling of `InterruptedException`

6.2 Best Practices Illustrated

- Implementing `Runnable` rather than extending `Thread` (allows extending other classes)
- Using `start()` instead of directly calling `run()`
- Assigning meaningful thread names for debugging
- Handling checked exceptions properly
- Keeping thread logic simple and focused

6.3 Skills Acquired

- Understanding the difference between threads and processes
- Creating and managing multiple threads in Java
- Recognizing concurrent vs. sequential execution
- Writing thread-safe basic code
- Debugging multi-threaded applications with thread names

---

7. Limitations and Future Enhancements

7.1 Current Limitations

- **No Error Handling:** Doesn't handle actual download failures
- **Simulated Delays:** Uses fixed 500ms sleep; real downloads vary
- **No Progress Tracking:** Doesn't actually download files
- **No Synchronization:** Doesn't use locks or synchronization
- **No Thread Control:** Can't pause, resume, or cancel downloads
- **No Real File I/O:** Purely console output simulation

7.2 Possible Enhancements

1. **File Download Service:** Implement actual file downloads using URLConnection or HttpClient
2. **Progress Callbacks:** Use callbacks or observers to report progress to UI
3. **Error Recovery:** Implement retry logic for failed downloads
4. **Thread Pool:** Use `ExecutorService` for efficient thread management
5. **Pause/Resume:** Add methods to pause and resume downloads
6. **Download Queue:** Manage multiple downloads with priority levels
7. **GUI Integration:** Create a graphical interface to show downloads
8. **Statistics:** Track download speeds, file sizes, time remaining
9. **Thread Synchronization:** Use locks for thread-safe operations on shared resources
10. **Timeout Handling:** Implement timeout mechanisms for stalled downloads

---

8. Advantages of This Implementation

- **Educational Value:** Clearly demonstrates multithreading fundamentals
- **Simplicity:** Easy to understand without advanced features
- **Modularity:** Clean separation between file download logic and thread management
- **Extensibility:** Easy to modify for real file downloads
- **Thread Safety:** Simple design avoids common threading issues for this use case
- **Observable:** Clear console output shows concurrent execution patterns

---

9. Comparison: Sequential vs. Concurrent

Sequential Approach (Without Multithreading)

Start download File1.pdf

- Wait 500ms × 5 = 2500ms total
  Complete File1
  Start download File2.mp4
- Wait 500ms × 5 = 2500ms total
  Complete File2
  Start download File3.zip
- Wait 500ms × 5 = 2500ms total
  Complete File3

Total Time: 7500ms (2.5 seconds)
Main Thread Status: BLOCKED throughout

Concurrent Approach (With Multithreading)

Start download File1.pdf (Thread-1) ———→ Complete (2500ms)
Start download File2.mp4 (Thread-2) ———→ Complete (2500ms) [Parallel]
Start download File3.zip (Thread-3) ———→ Complete (2500ms)

Total Time: ~2500ms (nearly identical to single download!)
Main Thread Status: FREE immediately to do other tasks

Performance Gain: 3× faster when running on multi-core systems.

10. Conclusion

This Download Manager Simulator project successfully demonstrates the core concepts of Java multithreading through a practical and easy-to-understand example[1][2]. By creating multiple threads to download files simultaneously, the project illustrates how concurrent execution improves application responsiveness and resource utilization.

Key Takeaways:

1. **Multithreading enables concurrent execution** of independent tasks within the same application
2. **The Runnable interface and `Thread` class** are the fundamental building blocks for multithreading
3. **Non-blocking behavior** allows the main thread to remain responsive while background tasks execute
4. **Thread names and proper exception handling** are essential for debugging and robustness
5. **Concurrent execution can significantly improve performance** for I/O-bound operations

This project provides a solid foundation for understanding more advanced threading concepts such as synchronization, thread pools, and inter-thread communication, which are essential for building scalable, responsive Java applications.

---

11. References

[1] Digital Ocean. (2025). Multithreading in Java: Concepts, Examples, and Best Practices. Retrieved from https://www.digitalocean.com/community/tutorials/multithreading-in-java

[2] GeeksforGeeks. (2025). Multithreading in Java. Retrieved from https://www.geeksforgeeks.org/java/multithreading-in-java/

[3] Oracle Java Documentation. Thread Class. Retrieved from https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

[4] Oracle Java Documentation. Runnable Interface. Retrieved from https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html

---

12. Appendices

Appendix A: Complete Source Code

FileDownloader.java
class FileDownloader implements Runnable {
private String fileName;

```
public FileDownloader(String fileName) {
    this.fileName = fileName;
}
```

```
    @Override
    public void run() {
        System.out.println("Starting download: " + fileName + " on " + Thread.current
        try {
            for (int i = 1; i <= 5; i++) {
                System.out.println("Downloading " + fileName + " ... " + (i * 20) + "% comp
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Download interrupted for: " + fileName);
        }
        System.out.println("Download completed: " + fileName + " on " + Thread.curr
    }
```

}

project.java

```
public class project {
public static void main(String[] args) {
System.out.println("Download Manager Started\n");
```

```
        Thread file1 = new Thread(new FileDownloader("File1.pdf"), "Thread-1");
        Thread file2 = new Thread(new FileDownloader("File2.mp4"), "Thread-2");
        Thread file3 = new Thread(new FileDownloader("File3.zip"), "Thread-3");

        file1.start();
        file2.start();
        file3.start();

        System.out.println("Main thread is free to do other tasks...\n");
    }
```

}

Appendix B: Glossary

- **Thread:** A lightweight unit of execution within a process
- **Runnable:** An interface that defines a task to be executed by a thread
- **Concurrent:** Running at the same time or appearing to run at the same time
- **Multithreading:** The ability of a program to execute multiple threads simultaneously
- **Thread Pool:** A group of pre-created threads ready to execute tasks
- **Synchronization:** Mechanism to control access to shared resources in multithreaded environments
- **Deadlock:** Situation where two or more threads are blocked indefinitely waiting for each other
- **Race Condition:** Unpredictable behavior when multiple threads access shared data without synchronization

---

End of Report