

Multithreading and Final Integration

Threads, the Event Dispatch Thread, SwingWorker, synchronized, and shipping v1.0

Lecture + Lab 9 + final demo

Branch: week12-threads-final

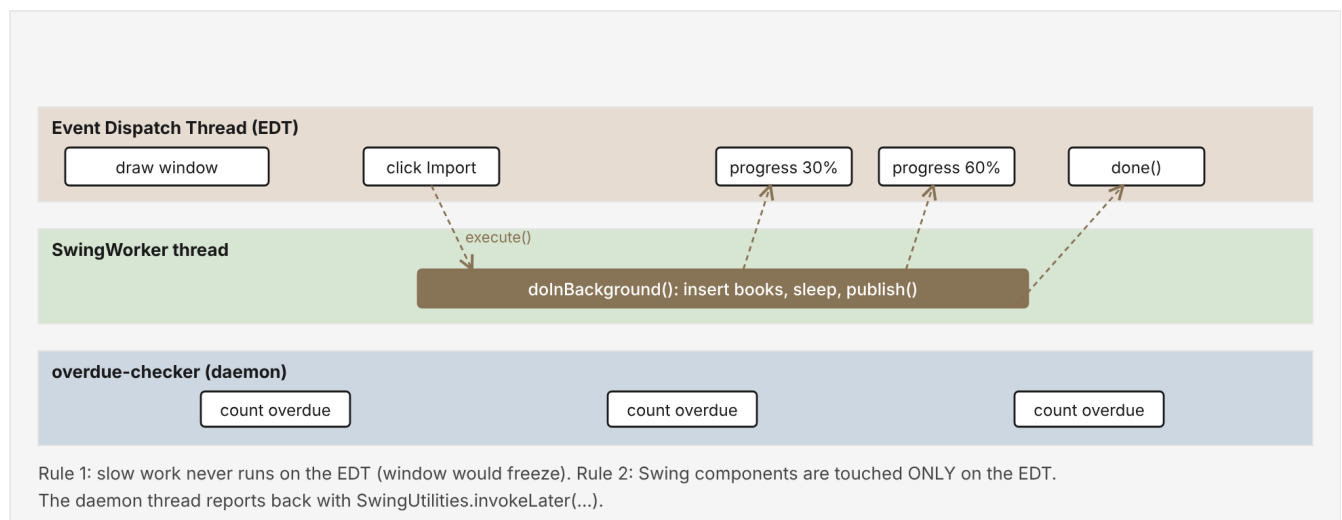
Learning outcomes 3, 4 and 5

What you will be able to do after today

- Create threads with `Runnable`, and explain daemon threads.
- Keep a GUI responsive with **SwingWorker**.
- Explain a **race condition** and fix it with `synchronized`.
- Package the LMS as a runnable `.jar`, tag **v1.0**, and demo it.

1. Why threads?

Press a button that does slow work on the **Event Dispatch Thread (EDT)** and the whole window freezes: no repainting, no clicks. The fix is to do slow work on **another thread** and send results back to the EDT.



Rule	
1	Slow work never runs on the EDT
2	Swing components are touched only on the EDT

2. Threads in plain Java

```
Thread t = new Thread(() -> System.out.println("hello from " + Thread.currentThread().getName()));
t.start();           // start(), NOT run(): run() would just call it on the current thread
```

A **daemon** thread will not keep the program alive. The LMS uses one to check for overdue loans every few seconds:

```

public static Thread start(LibraryController controller, long intervalMillis, IntConsumer onResult)
{
    Thread t = new Thread(new OverdueChecker(controller, intervalMillis, onResult), "overdue-
checker");
    t.setDaemon(true);
    t.start();
    return t;
}

```

It reports back using `SwingUtilities.invokeLater(...)` so that the label is updated **on the EDT**:

```

OverdueChecker.start(controller, 3000, count ->
    SwingUtilities.invokeLater(() -> overdueLabel.setText("Overdue: " + count)));

```

3. SwingWorker: background work with progress

```

public class ImportWorker extends SwingWorker<Integer, Integer> {
    @Override protected Integer doInBackground() throws Exception { // WORKER thread
        for (int i = 0; i < SAMPLE.length; i++) {
            controller.addItem("Book", id, title, author);
            Thread.sleep(400); // pretend it is slow
            publish((i + 1) * 100 / SAMPLE.length);
        }
        return added;
    }
    @Override protected void process(List<Integer> chunks) { // EDT
        bar.setValue(chunks.get(chunks.size() - 1));
    }
    @Override protected void done() { /* EDT: refresh the table */ }
}

```

Method	Runs on
<code>doInBackground()</code>	worker thread
<code>publish(x)</code>	called from the worker
<code>process(...)</code> , <code>done()</code>	EDT

While the import runs you can still select rows, because the EDT is free:

Library Management System

New item

Type: Book

ID:

Title:

Detail:

Add Item

Search: Search

ID	Title	Type	Detail	Status
B001	Clean Code	Book	Robert C. Martin	On loan: Nadeesha Perera
B002	Refactoring	Book	Martin Fowler	Available
D001	Introduction to Algorithms	DVD	120 min	Available
G001	Software Weekly	Magazine	Issue 42	Available

Library

New Member...

Loan Report

Loan History

Import Sample Books

B001 borrowed by M001

Overdue (>0 days): 1 30%

Member: M001 - Nadeesha Perera Borrow Return

Library Management System

New item

Type: Book

ID:

Title:

Detail:

Add Item

Search: Search

ID	Title	Type	Detail	Status
B001	Clean Code	Book	Robert C. Martin	On loan: Nadeesha Perera
B002	Refactoring	Book	Martin Fowler	Available
D001	Introduction to Algorithms	DVD	120 min	Available
G001	Software Weekly	Magazine	Issue 42	Available
I001	Design Patterns	Book	Erich Gamma	Available
I002	The Pragmatic Programmer	Book	Andrew Hunt	Available
I003	Code Complete	Book	Steve McConnell	Available
I004	Java Concurrency in Practice	Book	Brian Goetz	Available
I005	Domain-Driven Design	Book	Eric Evans	Available
I006	Clean Architecture	Book	Robert C. Martin	Available
I007	Head First Design Patterns	Book	Eric Freeman	Available
I008	Algorithms	Book	Robert Sedgewick	Available
I009	Operating System Concepts	Book	Abraham Silberschatz	Available
I010	Database System Concepts	Book	Abraham Silberschatz	Available

Library

New Member...

Loan Report

Loan History

Import Sample Books

Import finished

Overdue (>0 days): 1 Imported 10 book(s)

Member: M001 - Nadeesha Perera Borrow Return

4. Race conditions and synchronized

`count++` is **three steps**: read, add, write. Two threads can interleave and **lose updates**.

```
static volatile int unsafeCount = 0;    // two threads x 1,000,000 increments
static int safeCount = 0;
static synchronized void safeIncrement() { safeCount++; }
```

```
Expected      : 2000000
Unsynchronized: 1635764 <-- updates were lost
Synchronized  : 2000000
```

(The unsynchronised number changes every run, and some runs even get lucky. That is exactly what makes concurrency bugs dangerous.)

In the LMS, the table refresh (EDT) and the import (worker) both touch the `Library`. So its methods are `synchronized`, and so are the controller's compound operations (*check, then save, then add*), so that two threads cannot both pass the check for the same id.

```
public synchronized void borrowItem(String itemId, String memberId) throws ItemNotAvailableException
{ ... }
```

Tip `synchronized` = "one thread at a time on this object". Keep synchronised sections **short**: never hold the lock while sleeping or waiting for the network.

5. Final integration checklist

Area	Check
Git	Every earlier PR merged. <code>main</code> builds. Tag <code>v1.0</code> pushed.
Code quality	Meaningful names, no dead code, no <code>System.out</code> in the model
MVC	View uses controller only; DAOs hold all SQL
Errors	Every failure path shows a message, none crashes the app
Build	<code>mvn package</code> or <code>build.sh</code> produces a runnable jar
README	What it is, how to run, screenshot, architecture, your name and index
Repo	No <code>.class</code> , no <code>lms.db</code> , no secrets

Build and run:

```
mvn package
java -jar target/lms-1.0-shaded.jar
```

(Without Maven: put `sqlite-jdbc-3.46.0.0.jar` in `lib/`, run `./build.sh`, then `java -jar build/lms.jar`.)

Tag the release:

```
git tag v1.0
git push origin v1.0
```

6. The 5–10 minute demo

1. **(1 min)** What problem the LMS solves; show the UML.
2. **(3 min)** Live: add an item, add a member, borrow, trigger one error, return.
3. **(2 min)** Close and reopen to show the data persisted; show the history.
4. **(1 min)** Show the background import with the progress bar.
5. **(1 min)** One thing you are proud of and one thing you would improve.

7. Self-check

1. Why must `done()` and `process()` not be called from the worker thread by you?

2. What is the difference between `start()` and `run()` ?
3. Why does `count++` lose updates when two threads run it?
4. Why is a daemon thread right for the overdue checker?