

"Lab 9: Multithreading and Final Integration"

Keep the GUI responsive, fix race conditions, and ship a runnable v1.0

Lab 9 of 9

Branch: week12-threads-final

About 2 to 3 hours

Guide: LMS Guide, Step 9

Objective

Make the LMS **responsive** and **safe**: a background import with a progress bar, a daemon thread that checks for overdue loans, and **synchronized** access to shared data. Then **ship**: runnable jar, README, tag **v1.0**, and a 5 to 10 minute demo.

Before you start

- Lab 8 merged. Lecture 12 notes read.
- Final marks come from this PR plus the demo; plan time for the checklist at the end.

Git: start and submit

```
git checkout main
git pull origin main
git checkout -b week12-threads-final
```

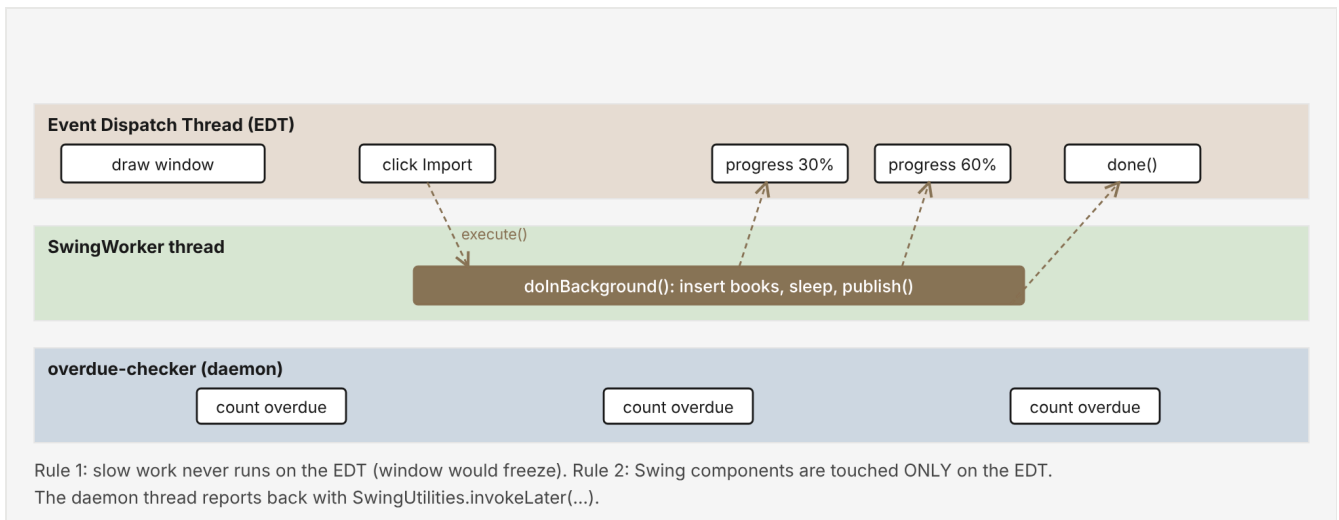
Commit small and often (see the *Git Support Guide*). When you finish:

```
git push -u origin week12-threads-final
```

Open a Pull Request titled **Week 12: Multithreading & Final Integration**, fill in the template, and request a review from your lecturer. After the review, merge it and tag **v1.0**.

Design first (10 minutes)

Draw a thread diagram for your app (docs/threads-week12.png):



Question to answer in your PR: *which thread updates the progress bar, and which thread does the slow work?*

Tasks

Task 1: see a race condition

Write `lms.RaceDemo`: two threads each increment a shared `static volatile int` 1,000,000 times; compare with a `synchronized` increment. Run it **five times** and paste all five results in the PR.

Task 2: `synchronized`

Make every `Library` method that touches the shared repositories `synchronized`, and the controller's compound methods (`addItem`, `addMember`, `borrow`, `giveBack`) too. Explain *check-then-act* in one sentence.

Task 3: `OverdueChecker`

A `Runnable` in `lms.service` that every `intervalMillis` calls `controller.countOverdue()` and passes the number to a callback. Start it as a **daemon** thread named `overdue-checker`. In the window, show `Overdue (>14 days): n` in the status area, updating the label with `SwingUtilities.invokeLater`.

Add `LoanDAO.countOverdue(int days)` (`WHERE returned_at IS NULL AND borrowed_at < ?`). Make the loan period configurable: `-Dlms.loan.days=0` makes every open loan overdue (handy for demos).

Task 4: `ImportWorker` extends `SwingWorker<Integer,Integer>`

An **Import Sample Books** button that adds ten books through the controller in `doInBackground()` (sleep 400 ms between them to simulate slowness), calls `publish(percent)`, updates a `JProgressBar` in `process`, and refreshes the table in `done()`. Disable the button while it runs. Skip ids that already exist.

While the import runs, select table rows to prove the window is not frozen.

Task 5: ship it

- `README.md`: what it is, features, how to run, screenshot, architecture (MVC and package list), your name and index number.

- Build: a `pom.xml` (with `sqlite-jdbc` and the shade plugin) **or** a `build.sh`. The result must start with `java -jar.`
- Clean up: no `System.out` left in the model, no unused imports, no commented-out code.
- Make sure **all earlier PRs are merged**, then tag the release:

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

Task 6: the demo (5 to 10 minutes)

Plan: problem and UML (1 min) → add, borrow, an error, return (3) → restart to show persistence and history (2) → background import (1) → what you are proud of and what you would improve (1).

Expected output

While the import runs, the progress bar moves and the table is still usable:

Library Management System

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

New item

Type: Book

ID:

Title:

Detail:

Library

B001 borrowed by M001

Overdue (>0 days): 1

30%

Member: M001 - Nadeesha Perera

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

RaceDemo should show lost updates in the unsynchronised case on most runs:

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

Checkpoints

Tick each before moving on. If a box stays empty, ask for help *now*; later steps build on it.

- ☐ RaceDemo lost updates on at least one of five runs
- ☐ The overdue label changes when a loan exists and `-Dlms.loan.days=0`
- ☐ The window never freezes during import
- ☐ `java -jar ...` starts the app on a **clean** copy of the repository (try it!)
- ☐ README complete; tag `v1.0` visible on GitHub
- ☐ All nine PRs merged

Experiments: break it on purpose

Learning happens when you predict, run, and compare. Write your **prediction** first, then paste the real compiler message or output into your PR description.

- Move the `Thread.sleep(400)` import loop into the button's `ActionListener` (on the EDT). Describe what you see.
- Update `overdueLabel.setText(...)` directly from the checker thread (without `invokeLater`). It may seem to work. Why is it still wrong?
- Remove `setDaemon(true)`. Close the window. What does the program do?
- Remove `synchronized` from `Library.addItem` and click **Import** while pressing **Search** repeatedly. Can you trigger an error?

Stretch tasks (optional, no marks lost if skipped)

- Use `ExecutorService` instead of a raw thread for the overdue check.
- Add a *Cancel* button to the import (`worker.cancel(true)`).
- Create a GitHub Release for `v1.0` and attach the jar.
- Add a unit test with JUnit for `Repository<T>`.

Common pitfalls

Symptom	Fix
GUI freezes during import	Slow code is on the EDT; move it to <code>doInBackground()</code>
<code>ConcurrentModificationException</code>	A list was changed while another thread looped over it; synchronise or copy
The jar says <code>no main manifest attribute</code>	Set <code>Main-Class: lms.GuiMain</code> in the manifest or shade plugin
<code>No suitable driver</code> when run from the jar	Include the SQLite jar (shade plugin) or <code>Class-Path</code> in the manifest
Tag not on GitHub	<code>git push origin v1.0</code> (tags are not pushed by default)

What to submit

- ☐ PR **Week 12: Multithreading & Final Integration** from branch `week12-threads-final`, merged
- ☐ Tag **v1.0**; README; runnable jar instructions
- ☐ Five `RaceDemo` outputs and the experiment results in the PR
- ☐ Demo in class (5 to 10 minutes)

Marking guide

Your PR is marked on four criteria. Each is scored 4 (excellent) to 1 (needs work) and weighted as shown.

Criterion (weight)	4	3	2	1
Works (40%)	All tasks run, expected output matches	Minor differences	Partly works	Does not compile or run
OOP design (30%)	Concepts of this week used correctly and deliberately	Mostly correct	Concepts present but misused	Concepts missing
Code quality (15%)	Clear names, small methods, no dead code, comments explain <i>why</i>	Mostly clean	Messy in places	Hard to read
Git and PR (15%)	Exact branch/PR name, small clear commits, full PR description, UML in docs/	One slip	Several slips	One giant commit or direct to <code>main</code>

Check Before you open the PR, read your own diff on GitHub as if you were the reviewer. Most deductions are things you would have spotted yourself.