Advanced Rust 2026 - Lab 2: Borrow-Checker-Driven API Design

Lukáš Hozda

Spring 2026

Lab Goals

1. Convert borrow-checker errors into API redesign decisions.
2. Practice owner/view split in realistic code.
3. Prepare for homework pair A2.

Time Plan (90 min)

1. 15 min - recap and diagnostic reading strategy
2. 25 min - exercise 1 (owner + views)
3. 25 min - exercise 2 (lifetime-aware iterator API)
4. 15 min - exercise 3 (minimize mutable borrow scope)
5. 10 min - wrap-up and Q&A

Exercise 1: Owner + View

Implement:

```
pub struct LineStore { /* owner */ }
pub struct LineView<'a> { /* borrowed view */ }
```

Requirements:

1. Store owns all line data.
2. Views borrow from store, no cloning.
3. Add method returning all non-empty lines as views.

Exercise 2: Lifetime-Aware Iterators

Implement iterator-returning API over borrowed data.

```
fn prefixed<'a>(&'a self, pfx: &'a str) → impl Iterator<Item = &'a str> + 'a
```

Constraints:

1. No allocation in iterator path.
2. Result lifetime tied to store and prefix.

Exercise 3: Short Borrow Refactor

Refactor intentionally failing code that holds ‘&mut self‘ too long.

Checklist:

1. Narrow mutable borrow scope.
2. Avoid unnecessary temporary clones.
3. Keep behavior identical.

Debrief

1. Which lifetime signatures communicate intent best?
2. Which borrow checker message was most useful and why?
3. How does this map to A2 homework pair?