The Matcher Trait: Code Companion

Reference code for the The Matcher Trait lecture. Sections correspond to the lecture document.

Section 1: What This File Does

```
/*!
This crate provides an interface for regular expressions, with a focus on line
oriented search. The purpose of this crate is to provide a low level matching
interface that permits any kind of substring or regex implementation to power
the search routines provided by the
['grep-searcher'](https://docs.rs/grep-searcher)
crate.

The primary thing provided by this crate is the ['Matcher'] trait. The trait
defines an abstract interface for text search. It is robust enough to support
everything from basic substring search all the way to arbitrarily complex
regular expression implementations without sacrificing performance.

A key design decision made in this crate is the use of *internal iteration*,
or otherwise known as the "push" model of searching. In this paradigm,
implementations of the 'Matcher' trait will drive search and execute callbacks
provided by the caller when a match is found.
*/
```

The module-level documentation establishes the crate's purpose and explicitly calls out the internal iteration design choice—a crucial architectural decision explained in the lecture.

Section 2: The Match Type - A Better Range

Note the #[derive(Copy)] —this is significant because std::ops::Range<usize> does not implement Copy. The Index implementations allow seamless integration with Rust's slice syntax.

Section 3: Line Terminators and Cross-Platform Reality

The private enum pattern (LineTerminatorImp) allows the internal representation to change without breaking API compatibility. Factory methods (byte(), crlf()) provide the only construction paths.

Section 4: ByteSet - Optimization Through Constraint

The bit vector representation uses exactly 32 bytes (4 × 8 bytes) to represent any subset of the 256 possible byte values. All operations are O(1) with no branching except for the bucket selection.

Section 5: The Captures Trait - Abstracting Group Extraction

```
replacement & u8
```

NoCaptures is a null object—it satisfies the trait contract but provides no functionality. This allows matchers without capture support to still work with APIs that expect the Captures associated type.

Section 6: The NoError Type - When Failure Is Impossible

```
/// NoError provides an error type for matchers that never produce errors.
///
/// The Display impl panics-if you see this error, you have a bug.
#[derive(Debug, Eq, PartialEq)]
pub struct NoError(()); // Empty struct, cannot be constructed externally

impl std::error::Error for NoError {
    fn description(*self) -> &str {
        "no error"
    }
}

impl std::fmt::Display for NoError {
    fn fmt(*self _ : &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        panic!("BUG for NoError: an impossible error occurred")
    }
}

impl From-NoError> for std::io::Error {
    fn from(_: NoError) -> std::io::Error {
        panic!("BUG for NoError: an impossible error occurred")
    }
}
```

The panicking implementations are intentional. NoError exists only to satisfy type system requirements—any actual attempt to use an error value indicates a programming mistake that should fail loudly.

Section 7: The Matcher Trait - Core Interface

```
-> Result<Option<Match> Self Error>;
-> Result<bool Self Error> {
```

Only find_at and new_captures are required. All other methods have default implementations that build on these two primitives.

Section 8: Internal Iteration Methods

The nested Result<Result<(), E>, Self::Error> return type distinguishes between matcher errors (outer) and callback errors (inner). The empty match handling prevents infinite loops on patterns like a*.

Section 9: Blanket Implementation for References

This blanket impl means functions accepting impl Matcher or M: Matcher automatically work with borrowed matchers. The #[inline] hints help eliminate the indirection cost.

Quick Reference

Key Types

Type Purpose

Match	Byte offset range with start <= end invariant
LineTerminator	
ByteSet	256-bit set of bytes (for optimization hints)
NoCaptures	Null implementation of Captures trait
NoError	

Matcher Trait Required Methods

```
fn find_at(&self, haystack: &[u8], at: usize) -> Result<Option<Match>,
    Self::Error>;
fn new_captures(&self) -> Result<Self::Captures, Self::Error>;
```

Matcher Associated Types

```
type Captures: Captures; // Use NoCaptures if unsupported
type Error: std::fmt::Display; // Use NoError if infallible
```

Optimization Hints

```
fn non_matching_bytes(&self) -> Option<&ByteSet>; // Bytes never in a match
fn line_terminator(&self) -> Option<LineTerminator>; // Line-oriented
optimization
fn find_candidate_line(&self, haystack: &[u8]) -> Result<Option<LineMatchKind>,
Self::Error>;
```

LineMatchKind Variants

```
LineMatchKind::Confirmed(usize) // Definite match at position
LineMatchKind::Candidate(usize) // Possible match, needs verification
```