Concurrent Cache Project

Learning Objectives

- LRU eviction with concurrent access
- Sharding to reduce lock contention
- TTL-based expiration
- Read-through vs write-through caching
- Cache stampede prevention

Requirements

Functional

1. **Get/Set operations** - Thread-safe read/write
2. **LRU eviction** - Remove least recently used when full
3. **TTL expiration** - Entries expire after configured time
4. **Max size cap** - Bounded memory usage

Non-Functional

1. **Performance** - >100k ops/sec
2. **Correctness** - No races (pass -race detector)
3. **Memory bounded** - Cleanup expired entries

Implementations

Naive (Single mutex, no eviction)

Problems: Global lock, memory leak, no TTL

Improved (Sharded, LRU, manual cleanup)

Fixes: 16 shards, LRU list, background cleanup

Final (Optimized sharding, metrics, tests)

Production: 256 shards, RWMutex, atomic metrics, comprehensive tests

Usage

```
cache := final.NewCache(final.Config{
    MaxSize: 10000,
    TTL: 5 * time.Minute,
    Shards: 256,
})

cache.Set("key", value)
if val, ok := cache.Get("key"); ok {
    // Use value
}
```

Success Criteria

- Understand sharding benefits (benchmark proof)
- Implement LRU correctly (MoveToFront on access)
- Handle TTL expiration (background cleanup)
- Write comprehensive tests (race, stress, accuracy)