Building Blocks for Concurrent Programming

Legacy Thread-Safe Containers

- Examples: `Vector`, `Hashtable`, and `Collections.synchronizedXxx` wrappers.
- **Pros**:
  - Simple thread safety
- **Cons**:
  - Poor scalability
  - Compound actions not atomic
  - Iterators require client-side locking

Iterator Limitations

- `ConcurrentModificationException` if mutation occurs during iterator usage.
- Locking during iteration can lead to performance bottlenecks or deadlock.
- Cloning the collection is an (expensive) alternative.

Concurrent Collections

- Modern scalable alternatives:
  - `ConcurrentHashMap`
  - `CopyOnWriteArrayList`
  - `ConcurrentLinkedQueue`
  - `ConcurrentSkipListMap`
- Features:
  - Fine-grained or lock-free algorithms
  - Weakly consistent iterators

How ConcurrentHashMap Works

- Non-blocking reads
- Lock striping for writes
- Iterators safe for concurrent use

CopyOnWrite Collections

- Best for many reads and few writes.
- Mutation copies array
- Iterators never fail
- Useful for observers/listeners

Blocking Queues

- Types:
  - `ArrayBlockingQueue`
  - `LinkedBlockingQueue`
  - `PriorityBlockingQueue`
  - `SynchronousQueue`
- Operations:
  - `put()` blocks if full
  - `take()` blocks if empty

Synchronizers

- `CountDownLatch`
- `CyclicBarrier`
- `Semaphore`
- `Exchanger`

Memoization

- Goals:
  - Avoid redundant expensive computations
  - Provide scalable caching

Summary

- Prefer concurrent collections over synchronized ones
- Blocking queues enable robust producer–consumer patterns
- Synchronizers simplify coordination
- Memoization provides scalable caching