Java Memory Model

What Is a Memory Model?

- Defines when writes by one thread become visible to another.
- Prevents incorrect assumptions due to CPU/compiler reordering.
- Ensures predictable multithreaded behavior.

Why We Need It

- Modern CPUs reorder instructions.
- Without synchronization, threads may see stale or partially built data.
- The Java Memory Model (JMM) provides rules and guarantees.

Platform Memory Models

- Multiprocessor systems have private CPU caches.
- Java hides platform-level memory model differences with JMM.
- JVM inserts memory barriers as needed.

Sequential Consistency vs Reality

- Sequential consistency: everything appears to occur in a global order.
- Real hardware does not guarantee sequential consistency.
- JMM allows reordering unless prevented by happens-before.

Reordering Code Example

```
static int x = 0, y = 0;
static int a = 0, b = 0;

Thread one = new Thread(() -> {
    a = 1;
    x = b;
});

Thread two = new Thread(() -> {
    b = 1;
    y = a;
});
```

Happens-Before

- The key visibility rule.
- If A happens-before B, B is guaranteed to observe A.
- No happens-before: reordering allowed.

Piggybacking on Synchronization

- Uses existing happens-before edges.
- Allows visibility without extra synchronization.
- Fragile and advanced technique.

Unsafe Publication

```
public class UnsafeLazyInit {
  private static Resource resource;

  public static Resource getInstance() {
      if (resource == null)
          resource = new Resource(); // unsafe
      return resource;
  }
}
```

- May reveal partially constructed objects.

Safe Publication Methods

- synchronized
- volatile
- static initializers
- thread-safe collections

Safe Lazy Initialization (Synchronized)

```
class SafeLazyInit {
    private static Resource resource;

    public static synchronized Resource getInstance() {
        if (resource == null)
            resource = new Resource();
        return resource;
    }
}
```

Initialization Safety

- Final fields have guaranteed visibility after construction.
- Objects reachable through final fields also safe.
- But escape during construction breaks guarantees.

Safe Immutable Example

```
class SafeStates {
    private final Map<String,String> states;

    public SafeStates() {
        states = new HashMap<>();
        states.put("alaska","AK");
        states.put("wyoming","WY");
    }

    public String get(String s) { return states.get(s); }
}
```

Summary

- JMM defines visibility and ordering guarantees.
- Happens-before provides structure for safe threading.
- Safe publication ensures fully visible objects.