Synchronization Policy

1. Purpose

The synchronization policy defines how concurrent threads within the **Bureaucracy System** coordinate access to shared resources to ensure data consistency, avoid race conditions, and maintain correct logical behavior when multiple customers and offices operate simultaneously

2. Concurrency Model

The system follows a **multi-threaded producer–consumer model**:

- Each **Customer** runs as an independent thread that requests documents in sequence
- Each **Office** runs in its own thread, serving customers from a blocking queue.
- The **BureaucracySystem** manages offices, documents, and random background events in separate threads
- Shared data structures include the maps of offices and documents, as well as office queues and document states

3. Thread Communication

- **Customer ↔ Office:**
  Offices issue documents by calling Customer.receiveDocument(), which wakes the customer waiting on that document (notifyAll())
- **System ↔ Office:**
  The main system thread and background controller may modify the office network (add new offices, restock paper). These operations are synchronized
- **Office internal threads**:
  Offices use local synchronization for paper stock management but rely on the thread-safe queue for handling customers

4. Deadlock and Race Condition Prevention

- Each synchronized section is short and non-nested, reducing the risk of deadlocks
- Shared collections are accessed in a consistent lock order (always through synchronized system methods)
- Thread-safe queues prevent blocking on producer–consumer interactions
- The design favors fine-grained synchronization, applied only to mutable shared states

5. Thread Lifecycle and Safety

- Offices and customers run independently and terminate naturally when the system stops
- The ExecutorService manages office threads efficiently
- The background controller thread is marked as a *daemon* to ensure graceful shutdown with the main program

6. Summary of Best Practices Applied

- Use of synchronized for state protection on shared mutable objects
- Use of concurrent collections (BlockingQueue) for thread-safe communication
- Use of wait() / notifyAll() for controlled synchronization between customer threads and document reception
- Avoidance of nested locks or long synchronized sections to maintain responsiveness