Transaction Management and Caching in Spring

This document explains the concepts of transaction management and caching in Spring, focusing on declarative and programmatic transaction management, as well as transaction propagation, isolation levels, and caching mechanisms.

1. Declarative Transaction Management

Declarative transaction management in Spring is achieved using the @Transactional annotation. This allows you to define transaction boundaries without manually managing the transaction lifecycle.

2. Programmatic Transaction Management

Programmatic transaction management gives you more flexibility but requires manual handling of
the transaction lifecycle. You can use the PlatformTransactionManager to manage transactions.

3. Transaction Propagation and Isolation Levels

Propagation defines how transactions relate to each other, while isolation levels determine the visibility of transaction changes to other transactions.

4. Caching in Spring

Caching is a mechanism to store frequently accessed data in memory to reduce the number of database hits and improves performance. Spring provides caching abstraction with annotations like
@Cacheable and @CacheEvict.

Example:

```
@Cacheable(value = "patients", key = "#patientId")
public Patient getPatientById(Long patientId) {
 return patientRepository.findById(patientId)
 .orElseThrow(() -> new IllegalArgumentException("Patient not found"));
}
@CacheEvict(value = "patients", key = "#patientId")
public void deletePatient(Long patientId) {
 patientRepository.deleteById(patientId);
}
```