@Transactional and @Cacheable

The `@Cacheable` and `@Transactional` annotations are both used to provide specific behaviors to methods in Spring applications:

1. **@Cacheable**: This annotation is used to indicate that the result of invoking a method should be cached. When the method is invoked with the same parameters, the cached result is returned instead of executing the method again. It's commonly used to improve performance by caching the results of expensive operations. Here's an example:

   ```java
   @Cacheable("books")
   public Book findBookById(long id) {
       // Method implementation
   }
   ```

   In this example, the `findBookById` method will be executed only once for a given `id`, and subsequent invocations with the same `id` will return the cached result without executing the method again.

2. **@Transactional**: This annotation is used to specify that a method should run within a transactional context. Transactions ensure that multiple database operations either all succeed or all fail, maintaining data integrity. The `@Transactional` annotation can be applied at both the class and method level. Here's an example:

   ```java
   @Service
   @Transactional
   public class BookService {

       @Autowired
       private BookRepository bookRepository;

       public void saveBook(Book book) {
           bookRepository.save(book);
       }
   }
   ```

   In this example, the `saveBook` method will run within a transaction, ensuring that the `bookRepository.save(book)` operation is atomic.

Both annotations provide important functionality for managing method behavior in Spring applications, allowing developers to easily handle caching and transactions.

The `value` attribute inside the `@Cacheable` annotation is used to specify the name of the cache in which the results of the annotated method should be stored or retrieved.

In Spring's caching abstraction, you typically define one or more cache managers, each of which manages one or more named caches. When you use `@Cacheable`, you specify the name of the cache you want to use for caching the results of the annotated method.

Here's an example:

```java
@Service
public class ProductService {

    @Cacheable(value = "productCache")
    public Product getProductById(Long id) {
        // This method will be cached under the "productCache" cache
        // The result will be stored in the cache based on the method parameters
        return productRepository.findById(id);
    }
}
```

In this example, the `@Cacheable(value = "productCache")` annotation specifies that the `getProductById` method results should be cached using the cache named "productCache". If the method is invoked with the same parameters again, Spring will first check if the result is already cached in the "productCache" cache and return it directly without executing the method again if found. Otherwise, it will execute the method and cache the result under the specified cache name.