Java Coding Question: Exception Handling with OOP Concepts

Problem Statement:

You are tasked with developing a small library management system in Java. The system should allow users to add books, lend books to members, and return books. The system should handle various exceptions that may occur during these operations, such as attempting to lend a book that is not available or returning a book that was never borrowed.

Requirements:

1. **Create a `Book` class** with the following attributes:
   - `String title`
   - `String author`
   - `boolean isLent` (indicates whether the book is currently lent out)
2. **Create a `Library` class** that maintains a list of `Book` objects and provides methods to:
   - Add a new book to the library.
   - Lend a book to a member.
   - Return a book.
3. **Create custom exception classes** for the following scenarios:
   - `BookNotFoundException`: Thrown when trying to lend or return a book that does not exist in the library.
   - `BookAlreadyLentException`: Thrown when trying to lend a book that is already lent out.
   - `BookNotLentException`: Thrown when trying to return a book that is not currently lent out.
4. **Implement the following methods in the `Library` class**:
   - `public void addBook(Book book)`: Adds a new book to the library.
   - `public void lendBook(String title) throws BookNotFoundException, BookAlreadyLentException`: Lends a book with the given title to a member. Throws `BookNotFoundException` if the book does not exist in the library and `BookAlreadyLentException` if the book is already lent out.
   - `public void returnBook(String title) throws BookNotFoundException, BookNotLentException`: Returns a book with the given title. Throws `BookNotFoundException` if the book does not exist in the library and `BookNotLentException` if the book is not currently lent out.
5. **Write a `main` method** to test your implementation. The `main` method should demonstrate the following:
   - Adding multiple books to the library.
   - Successfully lending and returning a book.
   - Attempting to lend a book that is already lent out and catching the `BookAlreadyLentException`.
   - Attempting to return a book that was never lent out and catching the `BookNotLentException`.
   - Attempting to lend or return a book that does not exist in the library and catching the `BookNotFoundException`.

Example:

```java
public class Main {
    public static void main(String[] args) {
        Library library = new Library();

        library.addBook(new Book("1984", "George Orwell"));
        library.addBook(new Book("To Kill a Mockingbird", "Harper Lee"));

        try {
            library.lendBook("1984");
            library.lendBook("1984"); // Should throw BookAlreadyLentException
        } catch (BookNotFoundException | BookAlreadyLentException e) {
            System.out.println(e.getMessage());
        }

        try {
            library.returnBook("1984");
            library.returnBook("1984"); // Should throw BookNotLentException
        } catch (BookNotFoundException | BookNotLentException e) {
            System.out.println(e.getMessage());
        }

        try {
            library.lendBook("The Great Gatsby"); // Should throw BookNotFoundException
        } catch (BookNotFoundException | BookAlreadyLentException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Expected Output:

```csharp
1984 has been lent out.
The book '1984' is already lent out.
1984 has been returned.
The book '1984' was not lent out.
The book 'The Great Gatsby' does not exist in the library.
```

Instructions:

1. Implement the `Book`, `Library`, and custom exception classes as described above.
2. Ensure that your methods handle exceptions appropriately, providing meaningful error messages.
3. Test your implementation using the `main` method to validate all scenarios.

Evaluation Criteria:

- Correctness of exception handling and custom exception implementation.
- Proper use of OOP concepts such as encapsulation and class design.
- Clarity and structure of code.
- Comprehensive testing of all possible exception scenarios.