SOFTWARE DESIGN AND ARCHITECTURE

LAB TASK 1

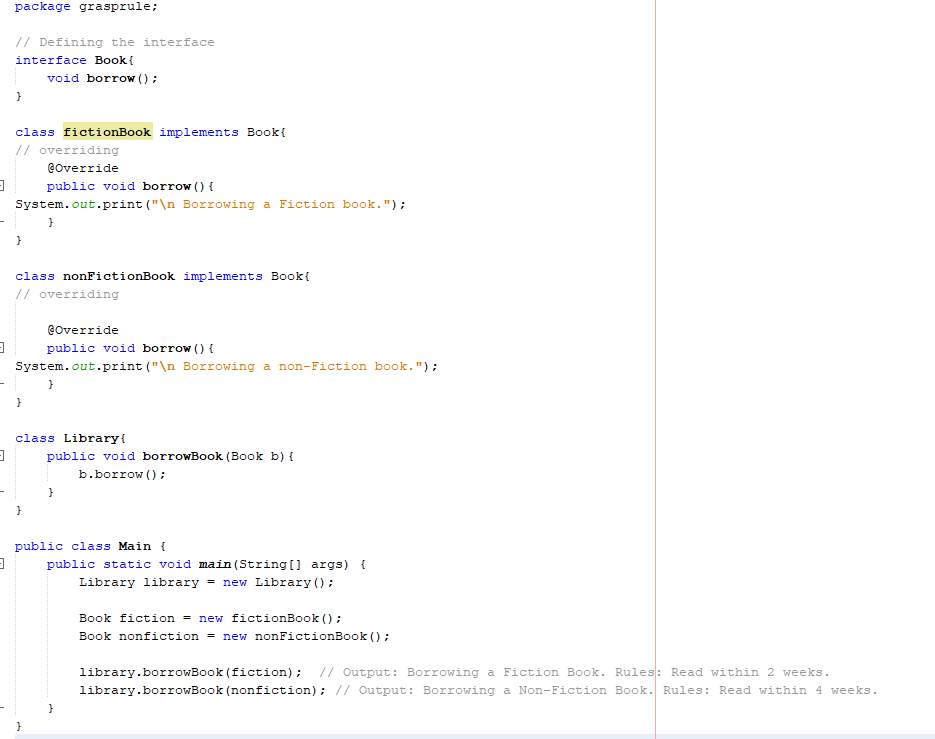
NAME: **USMAN ALI**

REG NO: **FA21-BSE-159**

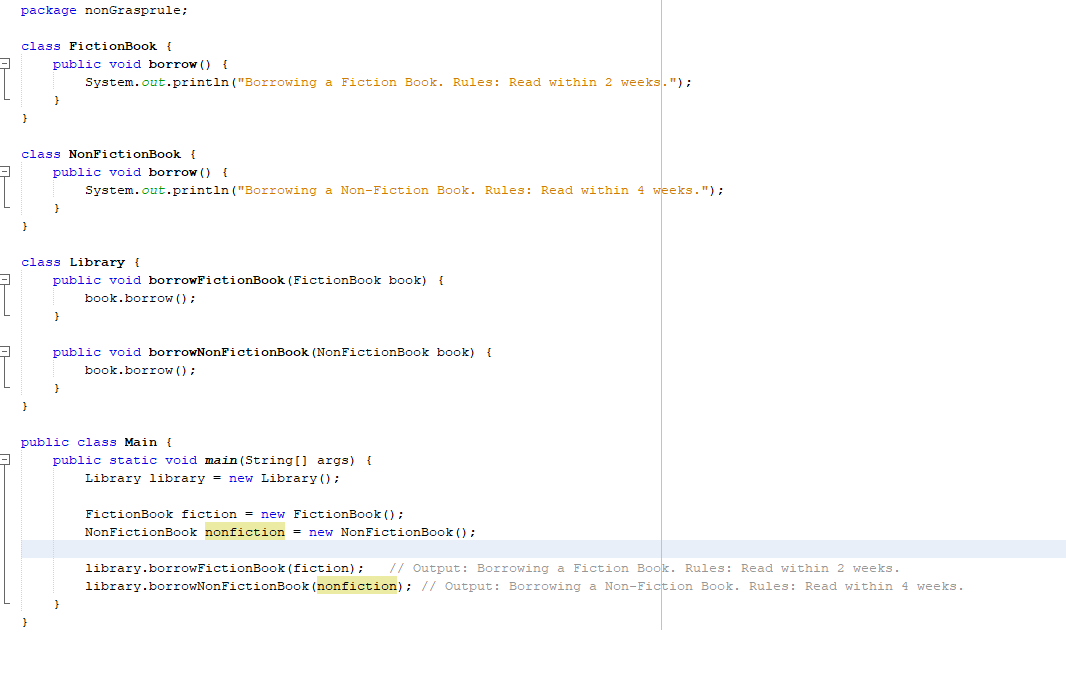
SUBMITTED TO: **SIR MUKHTIYAR ZAMIN**

Implementation of Polymorphism grasp principle with relevant coded example and attached video file.

With rule



Without rule



**1. Code Structure and Readability**

* **Concrete Classes**:
  + **Pros**: Simple and straightforward for small applications. Easy to understand when the number of book types is limited.
  + **Cons**: As the number of book types grows, it can lead to code duplication and harder maintenance.
* **Interface Approach**:
  + **Pros**: More organized, promotes code reuse, and reduces duplication. Any new book types can easily implement the Book interface.
  + **Cons**: Slightly more complex due to the interface, but this is generally manageable.

**2. Flexibility**

* **Concrete Classes**:
  + **Pros**: Good for small, fixed sets of classes.
  + **Cons**: Adding new book types requires modifying existing code, leading to potential errors.
* **Interface Approach**:
  + **Pros**: Very flexible; new book types can be added with minimal changes to existing code. Just implement the Book interface.
  + **Cons**: Requires more initial setup (defining interfaces), but pays off in larger applications.

**3. Performance**

* **Concrete Classes**:
  + Generally faster for small applications since there’s no overhead from interfaces or polymorphism.
* **Interface Approach**:
  + Slight overhead from dynamic method dispatch, but in most applications, this overhead is negligible. Modern JVM optimizations usually minimize performance differences.

**4. Maintainability**

* **Concrete Classes**:
  + Can become difficult to maintain as the application grows, especially with many similar classes.
* **Interface Approach**:
  + Easier to maintain as the codebase grows. New features and changes can be managed more effectively with interfaces.

**5. Testing**

* **Concrete Classes**:
  + Simple to test but might require multiple test cases if there are many book types.
* **Interface Approach**:
  + Easier to mock and test different implementations of the Book interface. You can write tests that focus on behavior rather than specific implementations.

**Conclusion**

In summary, while both approaches have their merits, using interfaces is generally more efficient in the long term, especially for larger and more complex systems. The interface approach enhances flexibility, maintainability, and testability, which are crucial factors for scalable applications. If you expect your library system to grow or require frequent changes, the interface-based design is the better choice.