**Hogwarts Archive – Design Report**

**Object-Oriented Structure**

The Hogwarts Archive system is centered around three main classes:

* **Archive** (controller/driver): Handles user commands, maintains collections of students and spellbooks, and directs interactions.
* **Student**: Represents each registered student, storing their ID, name, currently rented books, and rental history.
* **SpellBook**: Represents individual spellbooks, with metadata (serial number, title, inventor, type), availability, and rental history.

This separation follows the principle of **single responsibility**, ensuring each class manages its own state and behavior. The Archive class coordinates but does not store implementation details of individual students or spellbooks.

**Interactions Between Classes**

* **Student ↔ SpellBook**:  
  Students can rent or relinquish spellbooks. The rentBook and returnBook methods in Student, and the rentedBy field in SpellBook, provide bidirectional consistency. Each relinquish action updates both the student’s active rentals and the spellbook’s status.
* **Archive ↔ Student/SpellBook**:  
  The Archive class maintains maps (Map<Integer, Student> and Map<Integer, SpellBook>) for fast lookup. This ensures constant-time retrieval when handling commands such as RENT, RELINQUISH, and COMMON.

This design choice reflects the principle of **encapsulation**. These relationships are also clearly represented in the UML diagram, where Archive aggregates both Student and SpellBook, and a bidirectional association exists between Student and SpellBook through rental actions.

**Design Decisions**

1. **Unique IDs**:  
   Student IDs are auto-assigned starting from 100000, while spellbooks use their serial numbers. This ensures uniqueness and simplifies lookup operations.
2. **Rental History**:  
   Both students and spellbooks track histories. A student’s history enables queries like STUDENT HISTORY, while a spellbook’s history supports SPELLBOOK HISTORY. This duplication was intentional, as it reduces cross-lookups and matches the functional requirements.
3. **Case-Insensitive Queries**:  
   Queries like TYPE and INVENTOR are implemented using case-insensitive comparisons. This improves usability while maintaining internal consistency by storing canonical versions.
4. **Defensive Guards**:  
   Commands like RENT, RELINQUISH, and COMMON handle error cases explicitly (e.g., “No students in system”, “Unable to return spellbook”). This matches the assignment specification’s emphasis on **exact output** and improves robustness.
5. **Output Formatting**:  
   The outBlock helper ensures all outputs follow the required “user: …” prefix and block formatting. This centralizes formatting logic so that command methods focus purely on business logic.

**Object-Oriented Principles**

* **Encapsulation**: Each class hides its internal representation (e.g., Student’s currentBooks set is private and accessed only through methods).
* **Abstraction**: Commands like doRent and doRelinquish abstract away details of checking availability and updating histories.
* **Modularity**: Adding new commands (e.g., LIST POPULAR) would only require minimal changes to the Archive driver without affecting core classes.
* **Extensibility**: Future extensions (such as inheritance, e.g., specialized RestrictedSpellBook) could be introduced without disrupting existing logic.

**Conclusion**

The design balances **clarity, robustness, and extensibility through careful application of object-oriented principles such as encapsulation, abstraction, and modularity.** By structuring the system around Archive, Student, and SpellBook, each class is cohesive and testable. This ensures that the Hogwarts Archive can be easily maintained and extended to meet new requirements.