**Single Responsibility Principle (SRP)**

**Item**: Base class for all catalog items (books, magazines, CDs, etc.). Each subclass will handle specifics related to its type.

**User**: Base class for users. Different subclasses (Librarian, Patron) manage permissions and actions each can perform.

**Open/Closed Principle (OCP)**

**Catalog**: Manages items. It's designed to be extensible to support new item types without modifying existing code, using polymorphism.

**Liskov Substitution Principle (LSP)**

Subclasses of **Item** and **User** are designed to be substitutable for their base classes without affecting the system's behavior.

**Interface Segregation Principle (ISP)**

**ICheckoutable**, **IReservable**: Interfaces that segregate checkout and reservation functionalities, implemented by classes that require them.

**Dependency Inversion Principle (DIP)**

High-level modules like **CatalogManagement** interact with abstractions (**Item**, **User**) rather than concrete implementations.

**Implementation is on Assignment1.py file**

Explanation of SOLID Principles in Design:

**SRP**: **Item** and **User** act as base classes for specific types and actions, ensuring each class has a single responsibility.

**OCP**: New types of items can be added without modifying the existing **Catalog** class, demonstrating open for extension but closed for modification.

**LSP**: Subclasses of **Item** can be used in place of **Item** without altering the correctness of the program, e.g., in the **Catalog's** item list.

**ISP**: By using interfaces (though Python uses duck typing), we ensure that classes only implement what they need. This principle is more theoretical in this context but would apply in languages with explicit interfaces.

**DIP**: **CatalogManagement** and other high-level logic interact with abstract classes (**Item**, **User**) rather than concrete implementations, making the system more flexible and decoupled.