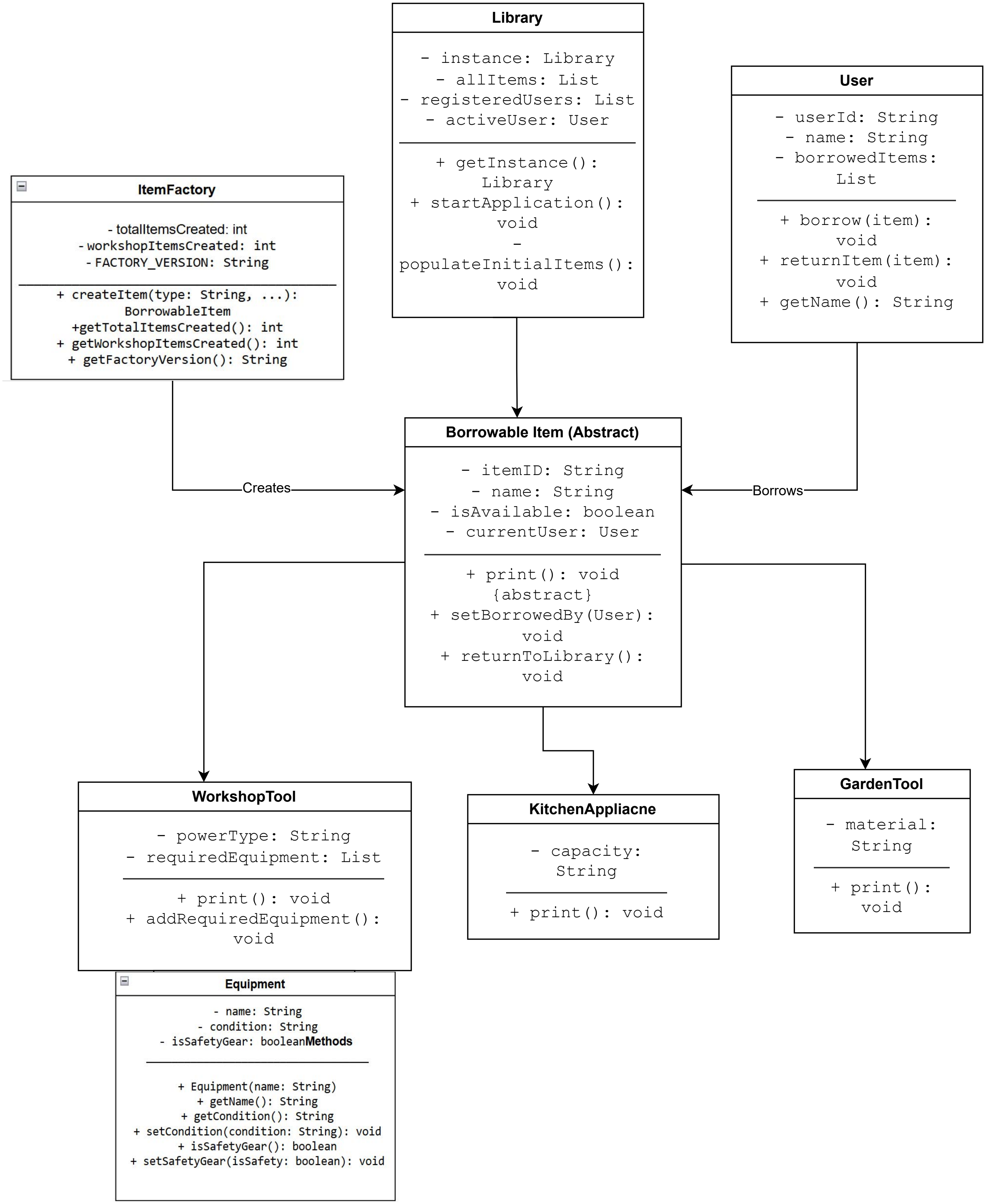
Sprint 2 Report: Library of Stuff

1. User Stories

These user stories define the functionality implemented in Sprint 2, focusing on the booking system and user interaction.

1. **View Inventory:** "As a user, I want to view all items in the library so that I can see what is available to borrow."
2. **Borrow Item:** "As a registered user, I want to borrow an available item by its ID so that I can use it for my project."
3. **Item Unavailability:** "As a user, I want to be informed if an item is already on loan so that I don't waste time trying to book it."
4. **Return Item:** "As a user, I want to return an item I currently possess so that others can use it."
5. **Loyalty Rewards:** "As a user, I want to earn loyalty points when I borrow items so that I am rewarded for frequent use of the library."

2. Class Diagram


Library
- instance: Library
- allItems: List
- registeredUsers: List
- activeUser: User
+ getInstance(): Library
+ startApplication(): void
- populateInitialItems(): void
User
- userId: String
- name: String
- borrowedItems: List
+ borrow(item): void
+ returnItem(item): void
+ getName(): String
ItemFactory
- totalItemsCreated: int
- workshopItemsCreated: int
- FACTORY_VERSION: String
+ createItem(type: String, ...): BorrowableItem
+getTotalItemsCreated(): int
+ getWorkshopItemsCreated(): int
+ getFactoryVersion(): String
Creates
Borrows
Borrowable Item (Abstract)
- itemID: String
- name: String
- isAvailable: boolean
- currentUser: User
+ print(): void {abstract}
+ setBorrowedBy(User): void
+ returnToLibrary(): void
WorkshopTool
- powerType: String
- requiredEquipment: List
+ print(): void
+ addRequiredEquipment(): void
KitchenAppliacne
- capacity: String
+ print(): void
GardenTool
- material: String
+ print(): void
Equipment
- name: String
- condition: String
- isSafetyGear: booleanMethods
+ Equipment(name: String)
+ getName(): String
+ getCondition(): String
+ setCondition(condition: String): void
+ isSafetyGear(): boolean
+ setSafetyGear(isSafety: boolean): void

3. Implementation of Design Patterns

To achieve a modular and robust architecture, I implemented two specific design patterns.

A. Singleton Pattern

- **Where it is used:** `src/Library.java`
- **Reason for implementation:** A real-world library is a single physical entity with one central inventory. If multiple instances of the `Library` class were created (e.g., `new Library()`), they would each have their own separate lists of items and users, leading to data inconsistencies.
- **How it works:** I made the `Library` constructor `private` and created a static `getInstance()` method. This ensures that only one instance of the `Library` object exists throughout the application's lifecycle, providing a global point of access to the inventory.

B. Factory Pattern

- **Where it is used:** `src/ItemFactory.java`
- **Reason for implementation:** As the application grows, adding new types of items (like "SportsEquipment") would require modifying the main logic with complex `if/else` statements. This violates the Open/Closed principle.
- **How it works:** I centralized the creation logic into the `ItemFactory` class. The `createItem` method takes a simple string type (e.g., "workshop") and handles the specific instantiation details (e.g., creating a `WorkshopTool`). This keeps the main `Library` code clean and decoupled from specific item classes.

4. Advanced Feature: Loyalty Point System

To meet the requirement for an additional "value-added" feature, I implemented a **Loyalty Points** system.

- **Functionality:** Every time a user successfully borrows an item, the system awards them **10 Loyalty Points**.
- **Implementation:**
  - The `User` class now has a `loyaltyPoints` attribute and a method `addLoyaltyPoints(int points)`.
  - The `Library` class calls this method inside the `borrowItem` logic upon a successful transaction.
  - The user's current point balance is displayed upon login and after every successful borrow, providing immediate positive feedback.

5. Technical Implementation Details

- **Encapsulation:** All class properties are private and accessed via getters/setters.
- **Inheritance:** `WorkshopTool`, `GardenTool`, and `KitchenAppliance` inherit from the abstract `BorrowableItem` class.
- **Polymorphism:** The `Library` class iterates through a list of `BorrowableItem` objects and calls `.print()`, which executes the specific behavior for each subclass.
- **Association & Aggregation:**
  - **Association:** The `BorrowableItem` class has a `currentUser` field, linking specific items to the user who borrowed them.
  - **Aggregation:** The `User` class maintains a `List<BorrowableItem>` representing the collection of items they currently hold.