Code Smell 1: Primitive Obsession

- **Location:** Primarily in `Item.java` (constructor and `setDimensions` method) and the `saveItem()` methods of `AddItemActivity.java` and `EditItemActivity.java`.
- **Description:** The code uses primitive types like `String` to represent more complex concepts such as item dimensions (`length`, `width`, `height`), despite the existence of a dedicated `Dimensions` class. This indicates a preference for simple data types over creating meaningful objects to encapsulate related data.
- **Why it is a problem:** This reduces code expressiveness, making it harder to read and understand. Validation logic for these attributes can become scattered instead of centralized within a single `Dimensions` class. It also leads to long parameter lists and obscures the natural relationships between data points.
- **Suggested Solution:** Refactor methods to directly use a `Dimensions` object within `Item`. Activities should create a `Dimensions` object from input data and pass this single object as a parameter, rather than individual strings.

Code Smell 2: Feature Envy

- **Location:** The `saveItem()` method in `AddItemActivity.java` and `EditItemActivity.java`.
- **Description:** The `saveItem` methods in these Activities exhibit "envy" towards the `Item` class. Instead of focusing solely on UI management, they are overly involved in meticulously constructing and configuring new `Item` objects (e.g., calling `new Item(...)`, then `setDimensions(...)`, `setStatus(...)`, `setBorrower(...)`). This suggests that business logic related to `Item` is misplaced within the View layer.
- **Why it is a problem:** This weakens the encapsulation of the `Item` class, as its creation/update logic leaks externally. It also leads to code duplication across Activities and violates the principle of separation of concerns (the View should not have deep knowledge of how the Model is constructed).
- **Suggested Solution:** Apply the "Move Method" refactoring technique. The logic for creating or updating an `Item` should be moved to the `ItemListController`. Activities would then simply call a single method on the Controller (e.g., `createItem()` or `updateItem()`) and pass the raw data collected from the UI.