Code Review

Design Critique

The application uses the Model-View-Controller (MVC) architecture and SOLID principles effectively in several key areas. Separate interfaces for command-line and GUI ensure flexibility for viewing options, and the Command Design Pattern is applied successfully in the controller, making the system modular and easy to extend. Dynamic command mapping via a function map further enforces the Open/Closed Principle, minimizing modification needs when adding new commands.

However, the command parsing logic relies on argument count rather than explicit argument patterns. This coupling of command syntax and logic makes future extensibility difficult and could cause incorrect dispatch as new and complicated commands are introduced.

Within the model layer, the interface decomposition is good, distinguishing calendar-level and event-level operations. Yet, event creation logic is duplicated across interfaces (*ICalendarModel* and *IExtendedZonedCalendarModel*), violating the Open/Closed Principle and making the code error prone. Introducing a single event creation strategy would reduce this risk.

Implementation Critique

The use of abstract base classes and builder patterns in the Event Model is commendable, increasing code reusability and immutability. The segregation between *SingleEvent* and *RecurringEvent* avoids unnecessary property duplication and makes the event hierarchy maintainable.

However, the controller has too much responsibility by directly invoking operations on the calendar model. This introduces tight coupling and suggests the need for a Facade pattern to mediate between controller logic and core models.

Further, formatting logic in the model layer (especially for CLI output) violates the MVC paradigm. Output concerns should reside solely in the View layer, enabling support for multiple presentation formats..

There is also redundant inheritance in the controller hierarchy, e.g., *ControllerWithView* extends *CLIController* but does not use its functionality. Such misuse of inheritance violates the Liskov Substitution Principle, and if needed composition would be a better fit here.

Finally, command implementations rely on switch-case statements based on argument length, which is error-prone. Refactoring this with a better parser would improve readability and robustness.

Documentation Critique

The JavaDocs are generally well-written and helpful for understanding class responsibilities and usage. Class and method-level documentation provides developers with enough information to navigate the codebase easily.

However, the README and USEME.md files lack design reasoning or explanations for specific architectural choices and how to use the application. Including this would enhance better understanding of the code.

The inconsistencies in the CSV import/export format, highlight the need for clearly documented data formats, especially when aiming for compatibility with external systems like Google Calendar.

Testing Critique

Testing across the application is incomplete and uneven. While unit tests exist for some components, particularly the model, there is minimal coverage for the text-based controller and no end-to-end integration testing.

There are mock classes, but these are either not used or replicate logic instead of abstracting behavior. For instance, the *MockController* reimplements controller logic instead of mocking it, failing to isolate the controller for effective testing. The view and model should be appropriately mocked for all types of controllers to allow comprehensive and independent testing.

Without proper testing, the application becomes error prone to enhancements and regressions, particularly in complex features like recurring event handling and conflict resolution.

Design/Code Strengths

- **Command Pattern & Dynamic Dispatch**: The controller's use of command mapping improves modularity and supports Open/Closed Principle.
- **Event Model Design**: Clean separation of single and recurring events; builder pattern ensures immutability and proper validation.
- **Interface Segregation**: Clear layering in calendar and event interfaces aligns with SOLID practices.
- **Decoupled GUI**: GUI components communicate through generalized callbacks, enabling multiple view integrations with minimal changes.
- **MVC Alignment**: Despite some minor violations, the overall MVC structure is strong and maintainable.

Design/Code Limitations and Suggestions

- **Command Parsing:** Refactor parsing logic to use argument keys or pattern matching instead of relying on input length. This will simplify extension and reduce error-proneness.
- **Model-Controller Coupling:** Introduce a Facade layer to abstract and mediate controller-model interactions, ensuring better separation and modularity.
- **Output Handling in Model:** Move all CLI or GUI output formatting logic from models into view classes to maintain strict MVC compliance.
- **Testing Deficiency:** Introduce comprehensive testing for all controllers, particularly the text interface. Use mocks correctly to isolate and test behavior.
- **View Logic Complexity:** Replace large switch blocks in GUI's *ActionListener* with action-command mappings or use the Command pattern for event handling.
- **View Interfaces:** Simplify the view layer by consolidating overly granular interfaces and promoting encapsulated behavior rather than exposing raw data.
- **Hard Coded values**: The application can make use of some constants.java to have static values at one place rather than having it hard coded everywhere.