EVENT MANAGEMENT

CSYE7374 50632 ST: Design Patterns SEC 01 - Group 5

Agni Chaturvedi
Khushank Mistry
Madhu Sai Kalyan Kaluri
Mit Sheth
Ruthvik Bangalore Ravi

PROBLEM STATEMENT

- Managing multiple event types with varying requirements becomes complex in traditional systems.
- Adding features like streaming, recording, or new sorting options often requires changing existing code, making the system harder to maintain and scale.
- Integration with external event sources is also challenging without a flexible architecture.

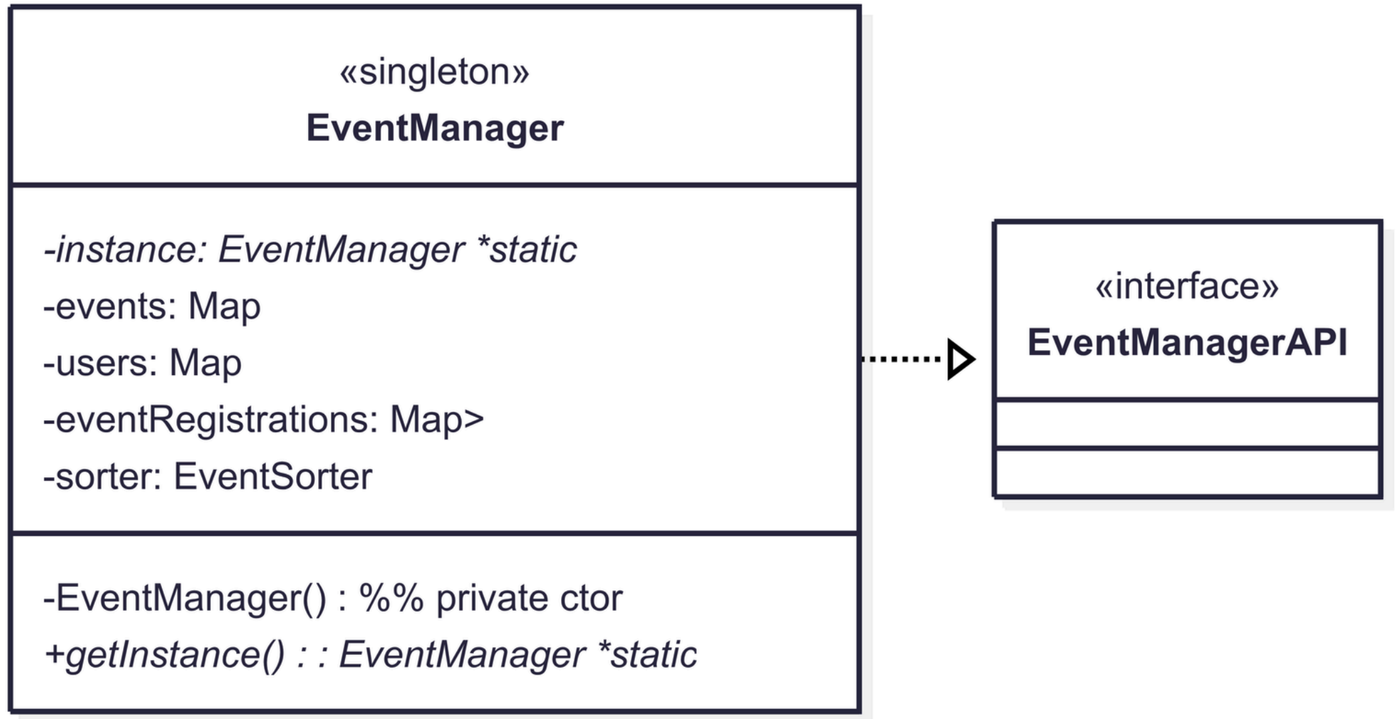
SOLUTION

- A modular Event Management System that supports creation, registration, management, and retrieval of events.
- Uses design patterns to keep the system extensible and maintainable.
- Allows addition of new features without changing core event classes.
- Supports sorting by multiple criteria and integrating third-party event sources.

SINGLETON DESIGN PATTERN

- **Centralized Event Management** - EventManager class implements Singleton pattern ensuring only one instance manages all events, users, and registrations across the entire campus event system

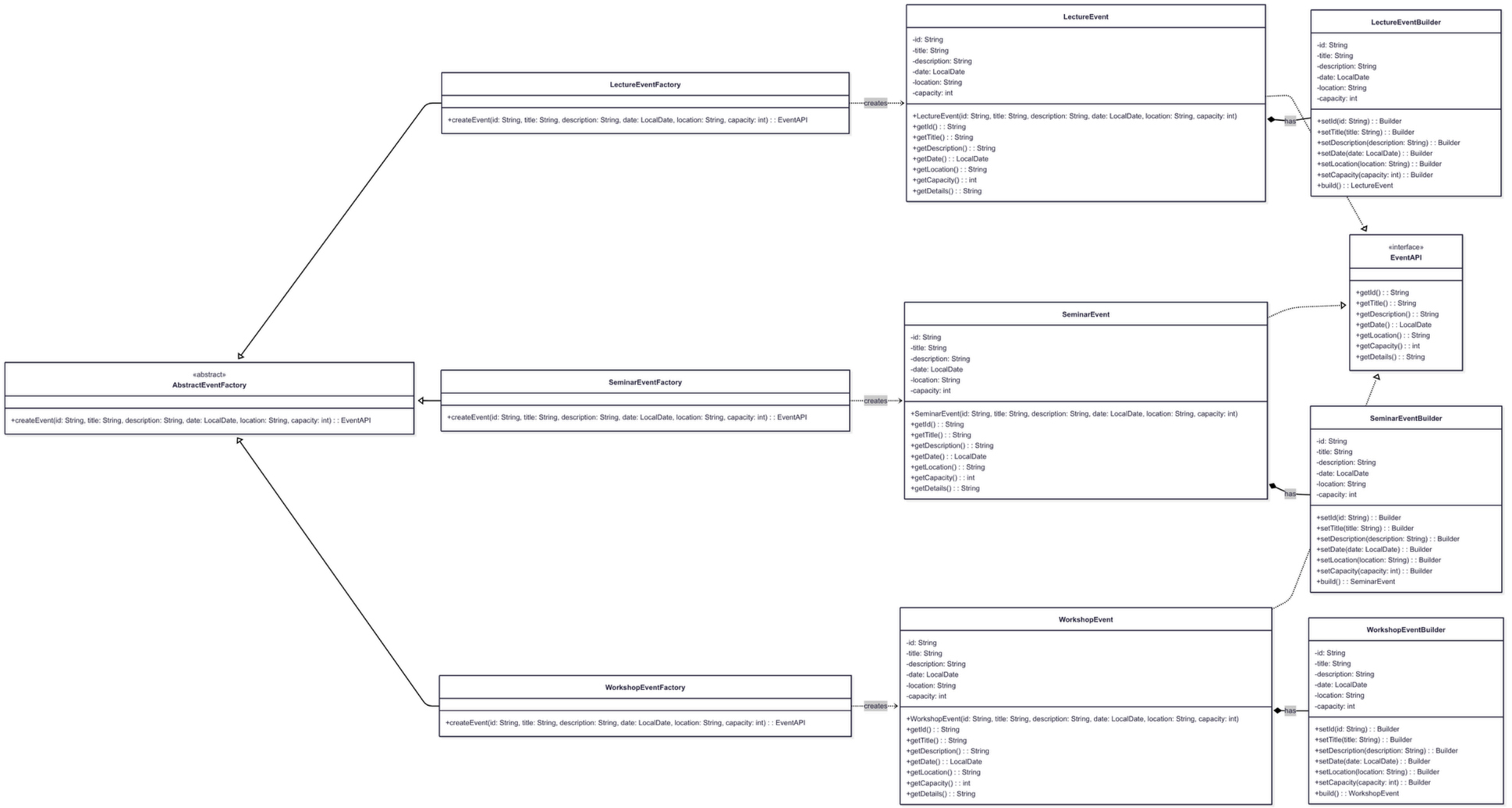
- **Global Access Control** - Single point of access through EventManager.getInstance() provides centralized data management, preventing duplicate instances and maintaining consistent system state for all event operations



ABSTRACT FACTORY

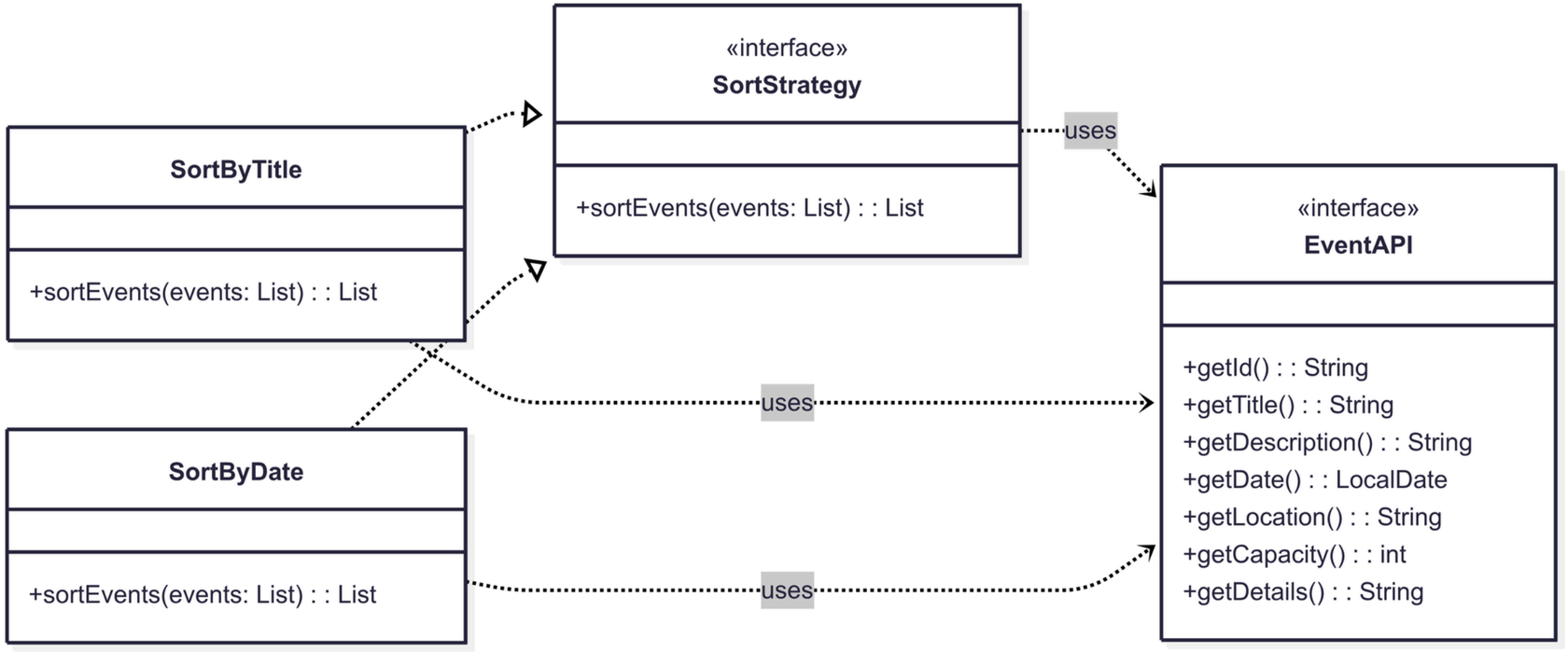
- **Abstract Factory for Event Creation** – AbstractEventFactory with its concrete factories (LectureEventFactory, SeminarEventFactory, WorkshopEventFactory) encapsulates the creation logic for different event types, ensuring consistent object construction across the system.
- **Product Specialization** – Each factory produces a specific event type implementing EventAPI (LectureEvent, SeminarEvent, WorkshopEvent), supporting scalability by allowing new event types to be added without altering existing creation logic.

ABSTRACT FACTORY



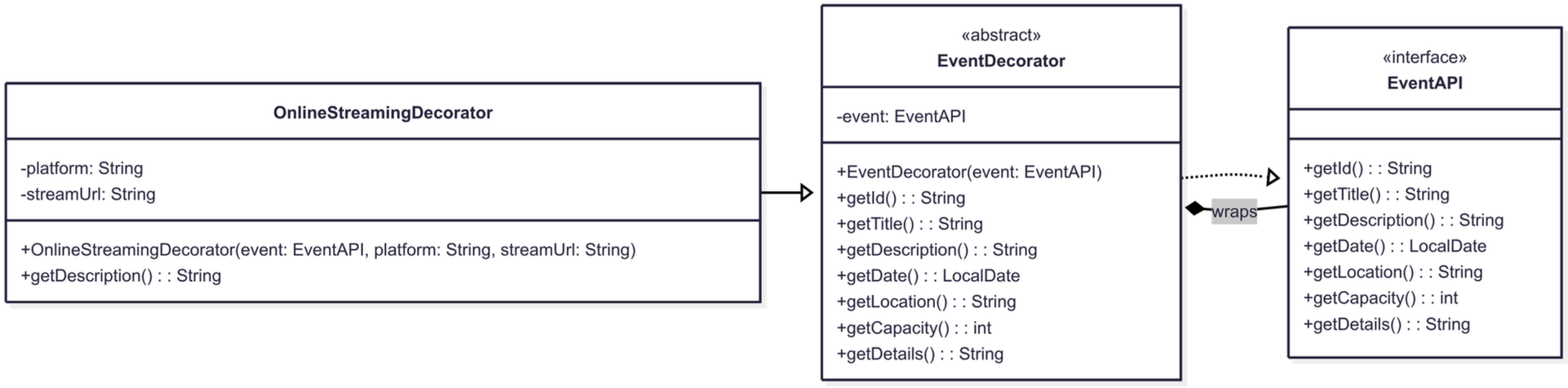
STRATEGY

- **Sorting Strategy Pattern** – SortStrategy interface defines a common contract for sorting events, with implementations like SortByTitle and SortByDate enabling flexible sorting criteria without modifying the core event management logic.
- **Runtime Behavior Switching** – The EventSorter can dynamically switch between different sorting strategies, empowering the system to adjust sorting order based on user preferences or specific business rules.



DECORATOR

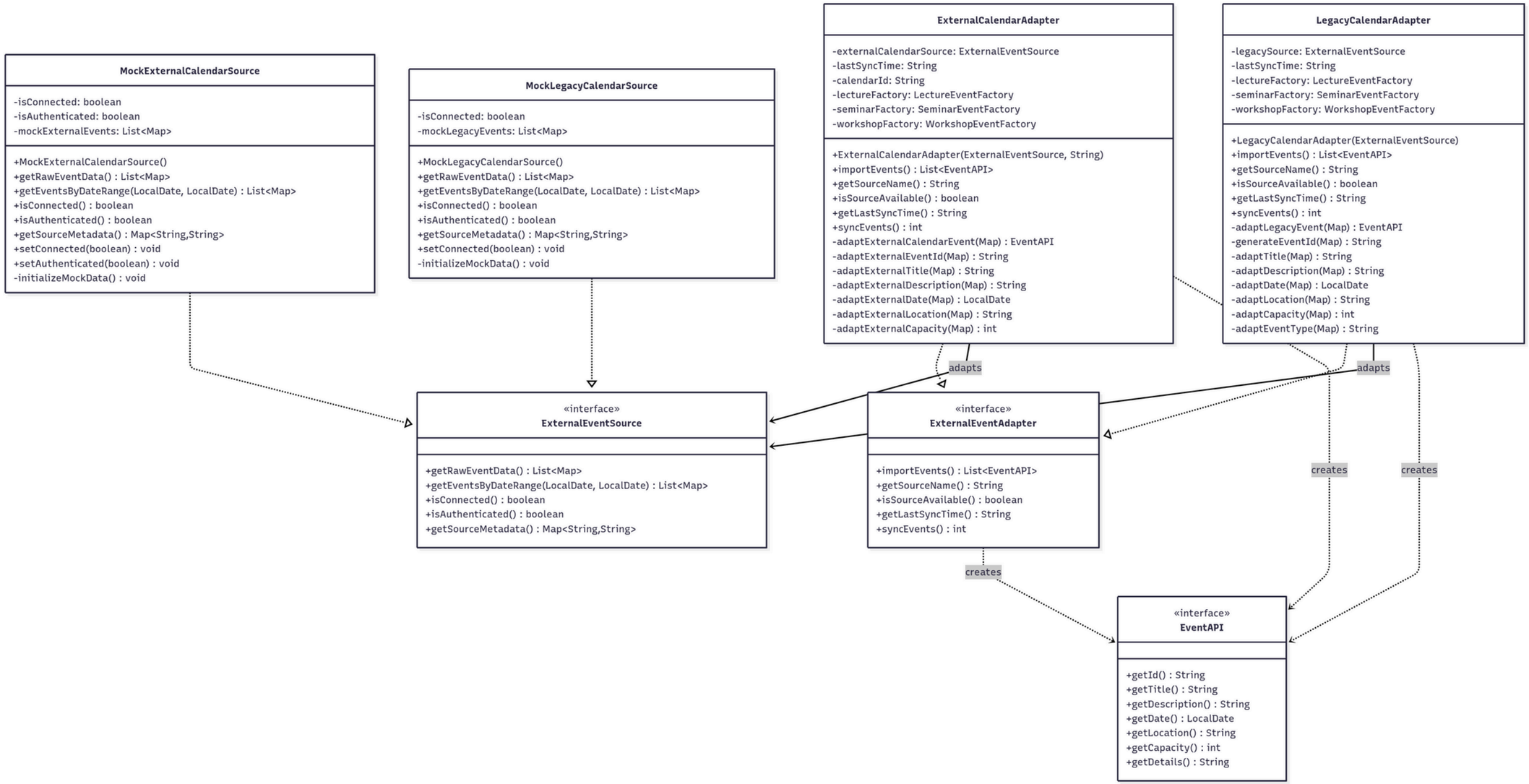
- **Decorator for Feature Enhancement** – EventDecorator abstract class wraps any EventAPI instance, delegating core methods while allowing selective method overrides for feature augmentation.
- **Dynamic Capability Extension** – OnlineStreamingDecorator enhances base event descriptions with streaming platform and URL details at runtime, enabling additional functionality without altering the original event classes.



ADAPTER

- **Interface Bridge for External Integration** – ExternalEventAdapter interface defines a unified contract for integrating diverse external calendar systems, enabling seamless data import while isolating system-specific implementation details from the core event management logic.
- **Format Translation and System Compatibility** – ExternalCalendarAdapter and LegacyCalendarAdapter transform incompatible external data formats (JSON structures, legacy field names, varied date formats) into standardized EventAPI instances, allowing the campus event system to consume events from multiple heterogeneous sources without modification to existing code.

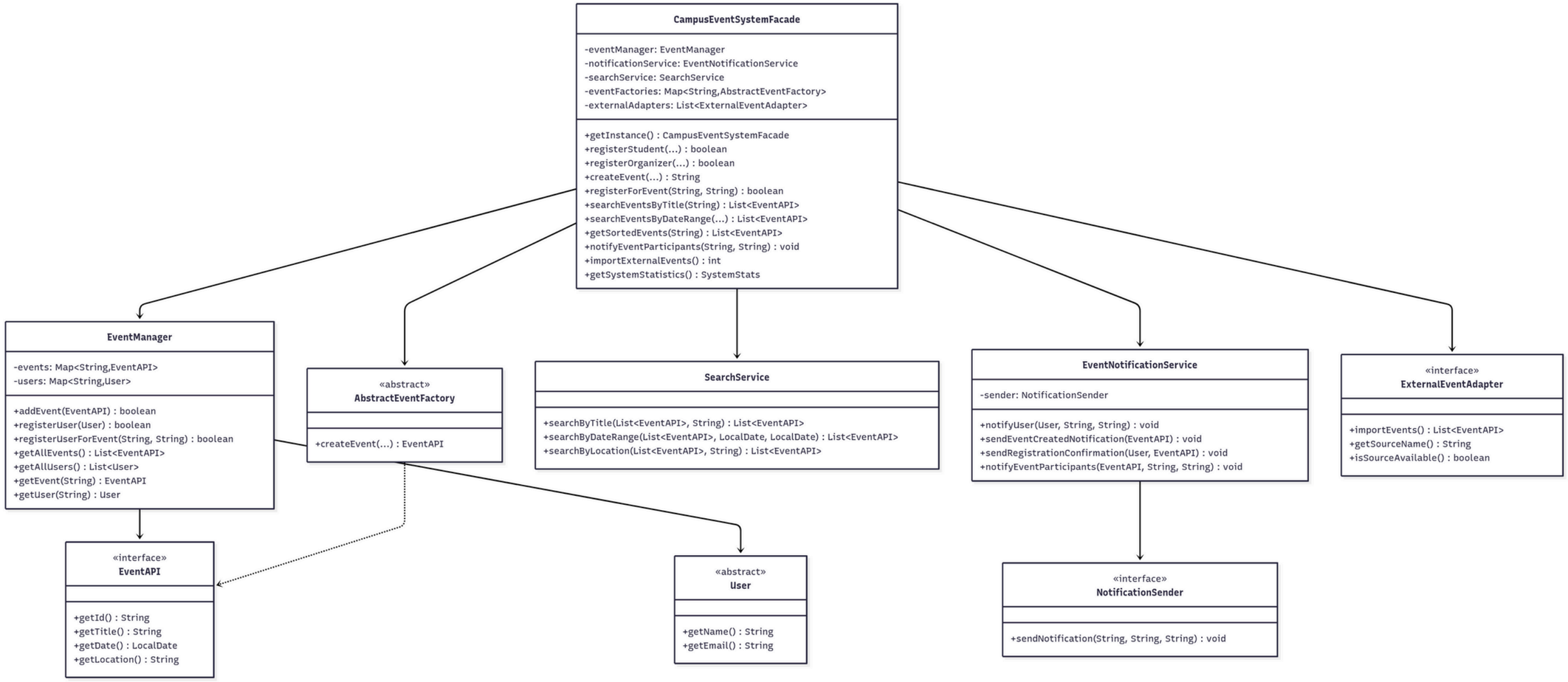
ADAPTER


MockExternalCalendarSource
-isConnected: boolean
-isAuthenticated: boolean
-mockExternalEvents: List<Map>
+MockExternalCalendarSource()
+getRawEventData() : List<Map>
+getEventsByDateRange(LocalDate, LocalDate) : List<Map>
+isConnected() : boolean
+isAuthenticated() : boolean
+getSourceMetadata() : Map<String,String>
+setConnected(boolean) : void
+setAuthenticated(boolean) : void
-initializeMockData() : void
MockLegacyCalendarSource
-isConnected: boolean
-mockLegacyEvents: List<Map>
+MockLegacyCalendarSource()
+getRawEventData() : List<Map>
+getEventsByDateRange(LocalDate, LocalDate) : List<Map>
+isConnected() : boolean
+isAuthenticated() : boolean
+getSourceMetadata() : Map<String,String>
+setConnected(boolean) : void
-initializeMockData() : void
ExternalCalendarAdapter
-externalCalendarSource: ExternalEventSource
-lastSyncTime: String
-calendarId: String
-lectureFactory: LectureEventFactory
-seminarFactory: SeminarEventFactory
-workshopFactory: WorkshopEventFactory
+ExternalCalendarAdapter(ExternalEventSource, String)
+importEvents() : List<EventAPI>
+getSourceName() : String
+isSourceAvailable() : boolean
+getLastSyncTime() : String
+syncEvents() : int
-adaptExternalCalendarEvent(Map) : EventAPI
-adaptExternalEventId(Map) : String
-adaptExternalTitle(Map) : String
-adaptExternalDescription(Map) : String
-adaptExternalDate(Map) : LocalDate
-adaptExternalLocation(Map) : String
-adaptExternalCapacity(Map) : int
LegacyCalendarAdapter
-legacySource: ExternalEventSource
-lastSyncTime: String
-lectureFactory: LectureEventFactory
-seminarFactory: SeminarEventFactory
-workshopFactory: WorkshopEventFactory
+LegacyCalendarAdapter(ExternalEventSource)
+importEvents() : List<EventAPI>
+getSourceName() : String
+isSourceAvailable() : boolean
+getLastSyncTime() : String
+syncEvents() : int
-adaptLegacyEvent(Map) : EventAPI
-generateEventId(Map) : String
-adaptTitle(Map) : String
-adaptDescription(Map) : String
-adaptDate(Map) : LocalDate
-adaptLocation(Map) : String
-adaptCapacity(Map) : int
-adaptEventType(Map) : String
adapts
adapts
«interface»
ExternalEventSource
+getRawEventData() : List<Map>
+getEventsByDateRange(LocalDate, LocalDate) : List<Map>
+isConnected() : boolean
+isAuthenticated() : boolean
+getSourceMetadata() : Map<String,String>
«interface»
ExternalEventAdapter
+importEvents() : List<EventAPI>
+getSourceName() : String
+isSourceAvailable() : boolean
+getLastSyncTime() : String
+syncEvents() : int
creates
creates
creates
«interface»
EventAPI
+getId() : String
+getTitle() : String
+getDescription() : String
+getDate() : LocalDate
+getLocation() : String
+getCapacity() : int
+getDetails() : String

FACADE

- **Unified System Interface for Complex Subsystem Coordination** – CampusEventSystemFacade provides a single, simplified entry point that orchestrates interactions between multiple complex subsystems (EventManager, EventNotificationService, SearchService, NotificationService) while hiding their intricate interdependencies from client code.
- **Streamlined Campus Event Operations** – The facade encapsulates complex workflows like user registration with welcome notifications, event creation with factory selection and participant alerts, and multi-criteria search with sorting strategies, transforming multi-step subsystem interactions into simple single-method calls for seamless campus event management.

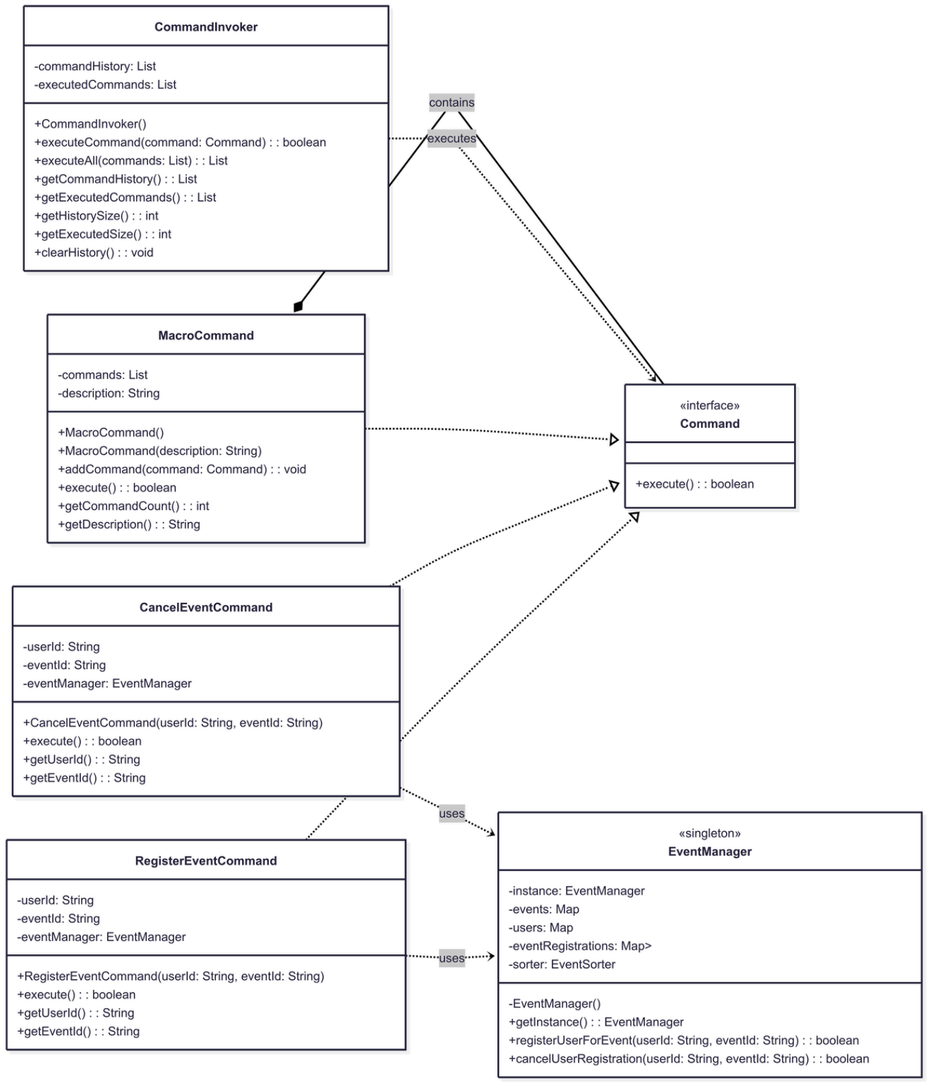
FACADE


CampusEventSystemFacade
-eventManager: EventManager
-notificationService: EventNotificationService
-searchService: SearchService
-eventFactories: Map<String,AbstractEventFactory>
-externalAdapters: List<ExternalEventAdapter>
+getInstance() : CampusEventSystemFacade
+registerStudent(...) : boolean
+registerOrganizer(...) : boolean
+createEvent(...) : String
+registerForEvent(String, String) : boolean
+searchEventsByTitle(String) : List<EventAPI>
+searchEventsByDateRange(...) : List<EventAPI>
+getSortedEvents(String) : List<EventAPI>
+notifyEventParticipants(String, String) : void
+importExternalEvents() : int
+getSystemStatistics() : SystemStats
EventManager
-events: Map<String,EventAPI>
-users: Map<String,User>
+addEvent(EventAPI) : boolean
+registerUser(User) : boolean
+registerUserForEvent(String, String) : boolean
+getAllEvents() : List<EventAPI>
+getAllUsers() : List<User>
+getEvent(String) : EventAPI
+getUser(String) : User
«abstract»
AbstractEventFactory
+createEvent(...) : EventAPI
SearchService
+searchByTitle(List<EventAPI>, String) : List<EventAPI>
+searchByDateRange(List<EventAPI>, LocalDate, LocalDate) : List<EventAPI>
+searchByLocation(List<EventAPI>, String) : List<EventAPI>
EventNotificationService
-sender: NotificationSender
+notifyUser(User, String, String) : void
+sendEventCreatedNotification(EventAPI) : void
+sendRegistrationConfirmation(User, EventAPI) : void
+notifyEventParticipants(EventAPI, String, String) : void
«interface»
ExternalEventAdapter
+importEvents() : List<EventAPI>
+getSourceName() : String
+isSourceAvailable() : boolean
«interface»
EventAPI
+getId() : String
+getTitle() : String
+getDate() : LocalDate
+getLocation() : String
«abstract»
User
+getName() : String
+getEmail() : String
«interface»
NotificationSender
+sendNotification(String, String, String) : void

COMMAND

- **Encapsulated Action Execution** – The Command interface defines a standard execute() method, allowing all user actions, such as event registration and cancellation, to be represented as discrete, reusable command objects.
- **Loose Coupling Between Sender and Receiver** – Concrete commands (RegisterEventCommand, CancelEventCommand) encapsulate the request parameters and delegate execution to the EventManager singleton, enabling changes to execution logic without modifying the calling code.
- **Centralized Command Orchestration** – The CommandInvoker acts as the command execution hub, triggering commands, tracking history, and maintaining a record of successful and attempted operations for auditing and rollback scenarios.

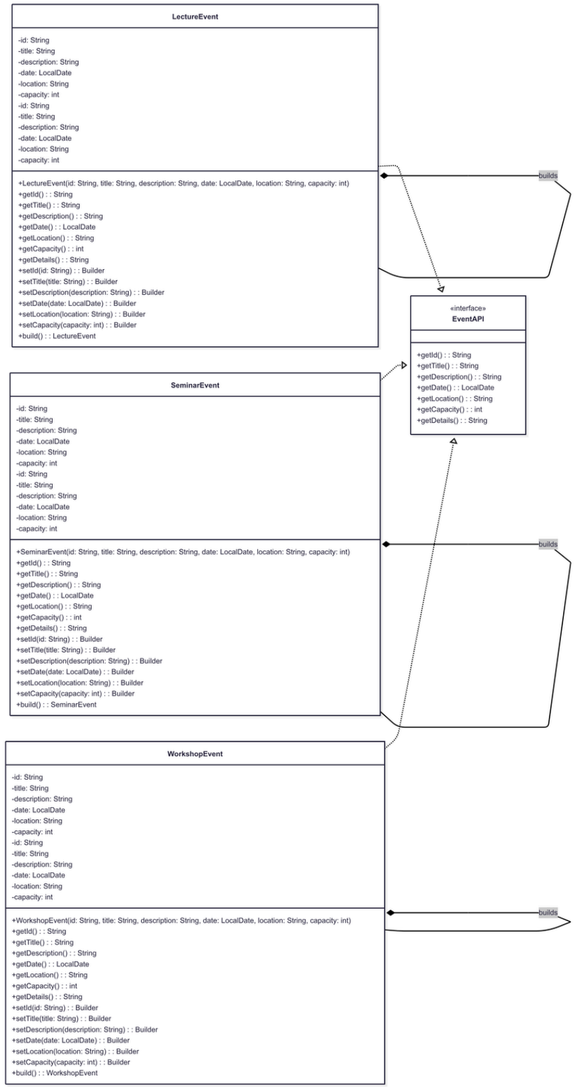
COMMAND


CommandInvoker
-commandHistory: List
-executedCommands: List
+CommandInvoker()
+executeCommand(command: Command) : : boolean
+executeAll(commands: List) : : List
+getCommandHistory() : : List
+getExecutedCommands() : : List
+getHistorySize() : : int
+getExecutedSize() : : int
+clearHistory() : : void
contains
executes
MacroCommand
-commands: List
-description: String
+MacroCommand()
+MacroCommand(description: String)
+addCommand(command: Command) : : void
+execute() : : boolean
+getCommandCount() : : int
+getDescription() : : String
«interface»
Command
+execute() : : boolean
CancelEventCommand
-userId: String
-eventId: String
-eventManager: EventManager
+CancelEventCommand(userId: String, eventId: String)
+execute() : : boolean
+getUserId() : : String
+getEventId() : : String
uses
RegisterEventCommand
-userId: String
-eventId: String
-eventManager: EventManager
+RegisterEventCommand(userId: String, eventId: String)
+execute() : : boolean
+getUserId() : : String
+getEventId() : : String
uses
«singleton»
EventManager
-instance: EventManager
-events: Map
-users: Map
-eventRegistrations: Map>
-sorter: EventSorter
-EventManager()
+getInstance() : : EventManager
+registerUserForEvent(userId: String, eventId: String) : : boolean
+cancelUserRegistration(userId: String, eventId: String) : : boolean

BUILDER

- **Fluent Object Construction** – Each event type (LectureEvent, SeminarEvent, WorkshopEvent) exposes an inner Builder with chainable setters and a build() method, enabling readable, step-by-step creation of complex event objects.
- **Immutability & Validation** – Builders collect and validate fields before instantiation; the built events are immutable (final fields), reducing bugs from partially constructed objects.

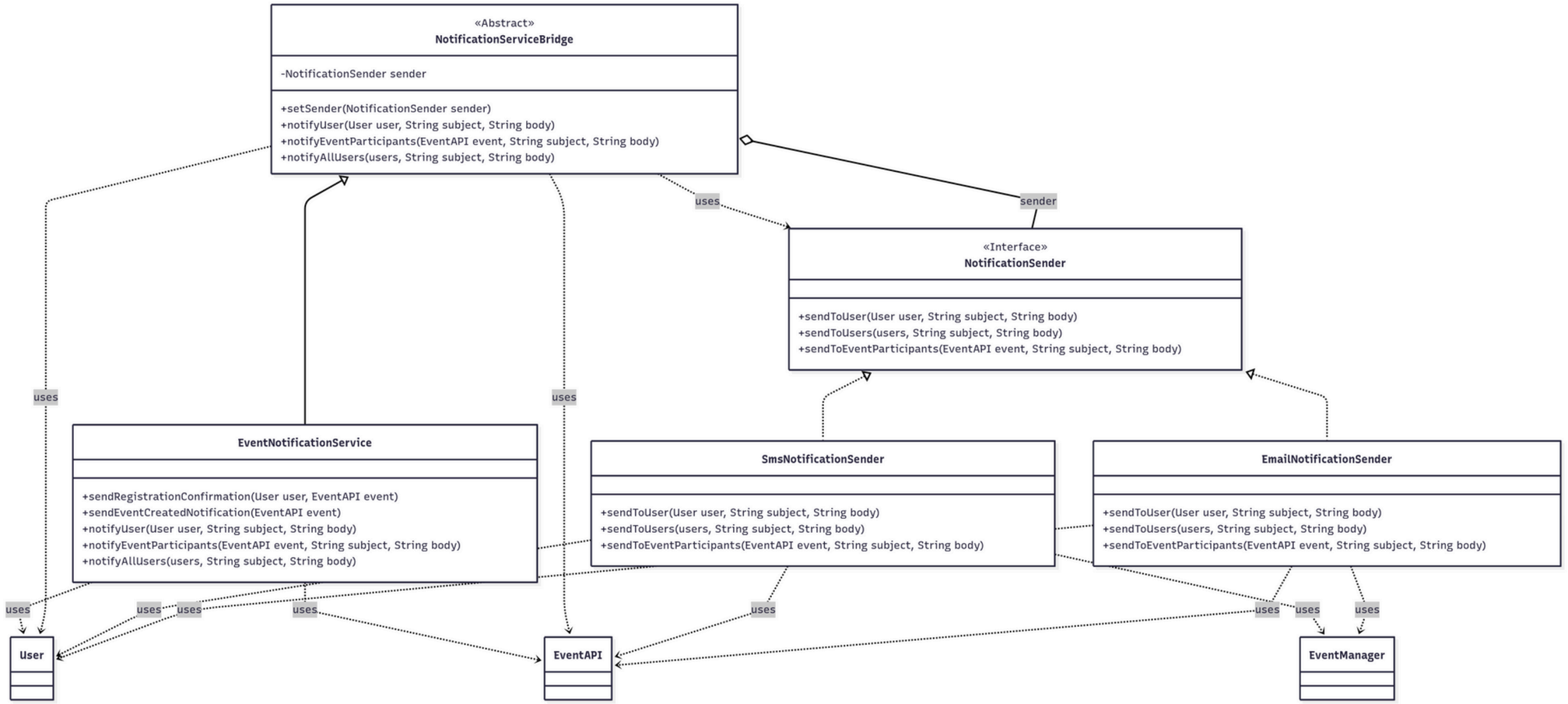
BUILDER


LectureEvent
-id: String
-title: String
-description: String
-date: LocalDate
-location: String
-capacity: int
-id: String
-title: String
-description: String
-date: LocalDate
-location: String
-capacity: int
+LectureEvent(id: String, title: String, description: String, date: LocalDate, location: String, capacity: int)
+getId() : : String
+getTitle() : : String
+getDescription() : : String
+getDate() : : LocalDate
+getLocation() : : String
+getCapacity() : : int
+getDetails() : : String
+setId(id: String) : : Builder
+setTitle(title: String) : : Builder
+setDescription(description: String) : : Builder
+setDate(date: LocalDate) : : Builder
+setLocation(location: String) : : Builder
+setCapacity(capacity: int) : : Builder
+build() : : LectureEvent
builds
«interface»
EventAPI
+getId() : : String
+getTitle() : : String
+getDescription() : : String
+getDate() : : LocalDate
+getLocation() : : String
+getCapacity() : : int
+getDetails() : : String
SeminarEvent
-id: String
-title: String
-description: String
-date: LocalDate
-location: String
-capacity: int
-id: String
-title: String
-description: String
-date: LocalDate
-location: String
-capacity: int
+SeminarEvent(id: String, title: String, description: String, date: LocalDate, location: String, capacity: int)
+getId() : : String
+getTitle() : : String
+getDescription() : : String
+getDate() : : LocalDate
+getLocation() : : String
+getCapacity() : : int
+getDetails() : : String
+setId(id: String) : : Builder
+setTitle(title: String) : : Builder
+setDescription(description: String) : : Builder
+setDate(date: LocalDate) : : Builder
+setLocation(location: String) : : Builder
+setCapacity(capacity: int) : : Builder
+build() : : SeminarEvent
builds
WorkshopEvent
-id: String
-title: String
-description: String
-date: LocalDate
-location: String
-capacity: int
-id: String
-title: String
-description: String
-date: LocalDate
-location: String
-capacity: int
+WorkshopEvent(id: String, title: String, description: String, date: LocalDate, location: String, capacity: int)
+getId() : : String
+getTitle() : : String
+getDescription() : : String
+getDate() : : LocalDate
+getLocation() : : String
+getCapacity() : : int
+getDetails() : : String
+setId(id: String) : : Builder
+setTitle(title: String) : : Builder
+setDescription(description: String) : : Builder
+setDate(date: LocalDate) : : Builder
+setLocation(location: String) : : Builder
+setCapacity(capacity: int) : : Builder
+build() : : WorkshopEvent
builds

BRIDGE

- Decouple event abstraction from notification implementation so both can vary independently.
- **Abstraction** (EventNotificationSender) – defines high-level interface to send notifications.
- **Implementors** (EmailSender, SMSSender, PushNotificationSender) – concrete ways of sending.
- **Concrete Abstractions** (EventNotification) – use the implementors without depending on their details.
- Can add new notification channels or change sending logic without modifying event classes.

BRIDGE


«Abstract»
NotificationServiceBridge
-NotificationSender sender
+setSender(NotificationSender sender)
+notifyUser(User user, String subject, String body)
+notifyEventParticipants(EventAPI event, String subject, String body)
+notifyAllUsers(users, String subject, String body)
uses
sender
«Interface»
NotificationSender
+sendToUser(User user, String subject, String body)
+sendToUsers(users, String subject, String body)
+sendToEventParticipants(EventAPI event, String subject, String body)
uses
uses
EventNotificationService
+sendRegistrationConfirmation(User user, EventAPI event)
+sendEventCreatedNotification(EventAPI event)
+notifyUser(User user, String subject, String body)
+notifyEventParticipants(EventAPI event, String subject, String body)
+notifyAllUsers(users, String subject, String body)
SmsNotificationSender
+sendToUser(User user, String subject, String body)
+sendToUsers(users, String subject, String body)
+sendToEventParticipants(EventAPI event, String subject, String body)
EmailNotificationSender
+sendToUser(User user, String subject, String body)
+sendToUsers(users, String subject, String body)
+sendToEventParticipants(EventAPI event, String subject, String body)
uses
uses
uses
uses
uses
uses
uses
uses
User
EventAPI
EventManager

DEMO

CONCLUSION

- Add user authentication and role-based permissions (organizer, attendee).
- Implement payment gateway integration for paid events.
- Build a web or mobile front-end for user-friendly interaction.
- Add analytics dashboard for event statistics (attendance, engagement).
- Add analytics dashboard for event statistics (attendance, engagement)

THANK YOU