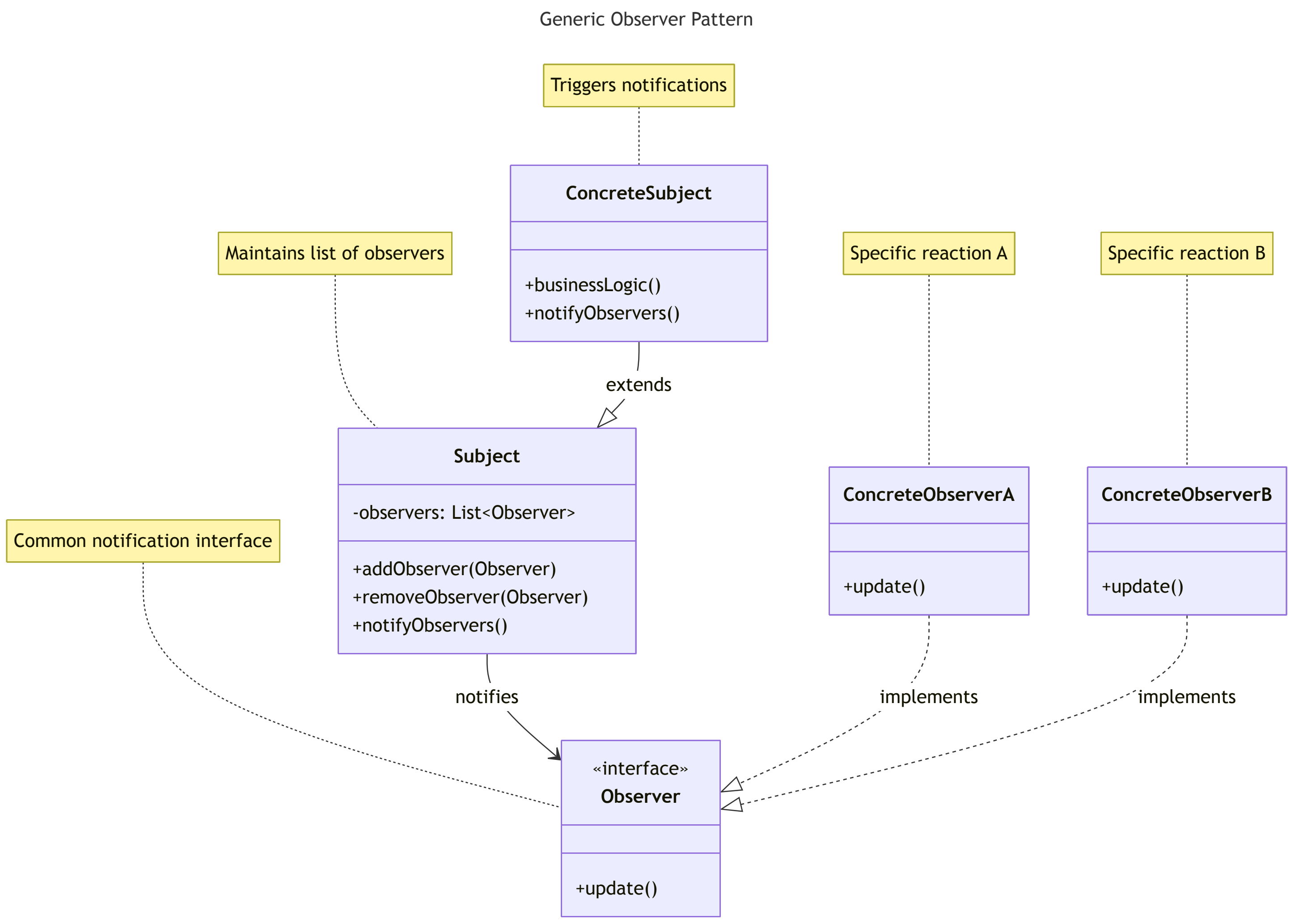
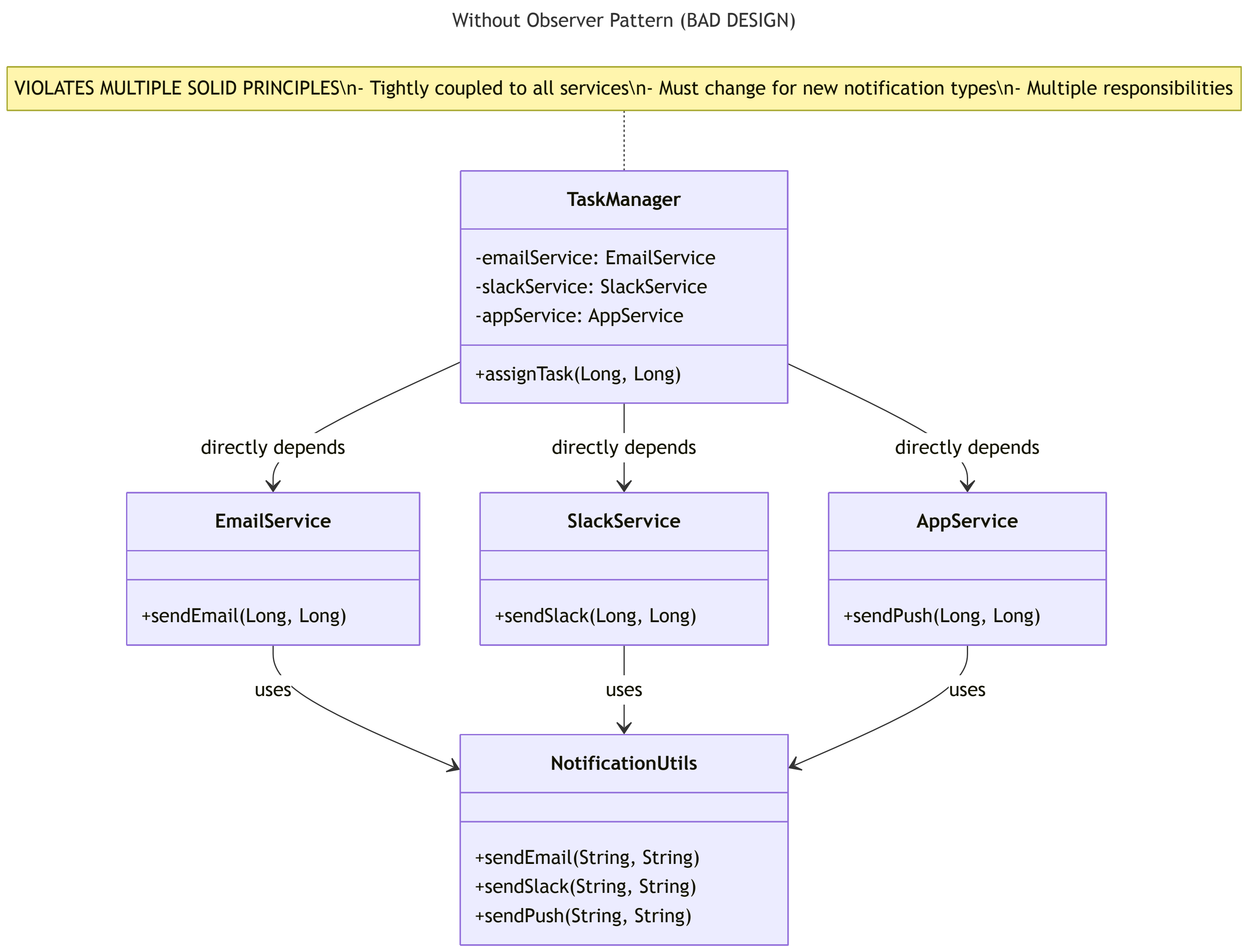
Generic Diagram



Observer Pattern for Task Management Application

Without Observer Pattern


Without Observer Pattern (BAD DESIGN)
VIOLATES MULTIPLE SOLID PRINCIPLES\n- Tightly coupled to all services\n- Must change for new notification types\n- Multiple responsibilities
TaskManager
-emailService: EmailService
-slackService: SlackService
-appService: AppService
+assignTask(Long, Long)
directly depends
directly depends
directly depends
EmailService
+sendEmail(Long, Long)
SlackService
+sendSlack(Long, Long)
AppService
+sendPush(Long, Long)
uses
uses
uses
NotificationUtils
+sendEmail(String, String)
+sendSlack(String, String)
+sendPush(String, String)

```java
// BAD - Without Observer Pattern
public class TaskManager {

    private EmailService emailService = new EmailService();
    private SlackService slackService = new SlackService();
    private AppService appService = new AppService();

    public void assignTask(Long taskId, Long userId) {
        emailService.sendEmail(userId, taskId); // ❌ Direct coupling
        slackService.sendSlack(userId, taskId); // ❌ Direct coupling
        appService.sendPush(userId, taskId);    // ❌ Direct coupling
        // Add new notification = modify this method ❌
    }
}

// Service classes are not standardized
public class EmailService {
    public void sendEmail(Long id, Long taskId) { /* implementation */ }
}

public class SlackService {
    public void sendSlack(Long id, Long taskId) { /* implementation */ }
}

public class AppService {
    public void sendPush(Long id, Long taskId) { /* implementation */ }
}
```

SOLID Principles Violated Without Observer Pattern

🔴 Single Responsibility Principle (SRP)

- TaskManager handles task assignment + manages all notification services
- One class responsible for multiple notification concerns

🔴 Open/Closed Principle (OCP)

- Must modify TaskManager to add new notification services
- Not open for extension, requires modification

🔴 Dependency Inversion Principle (DIP)

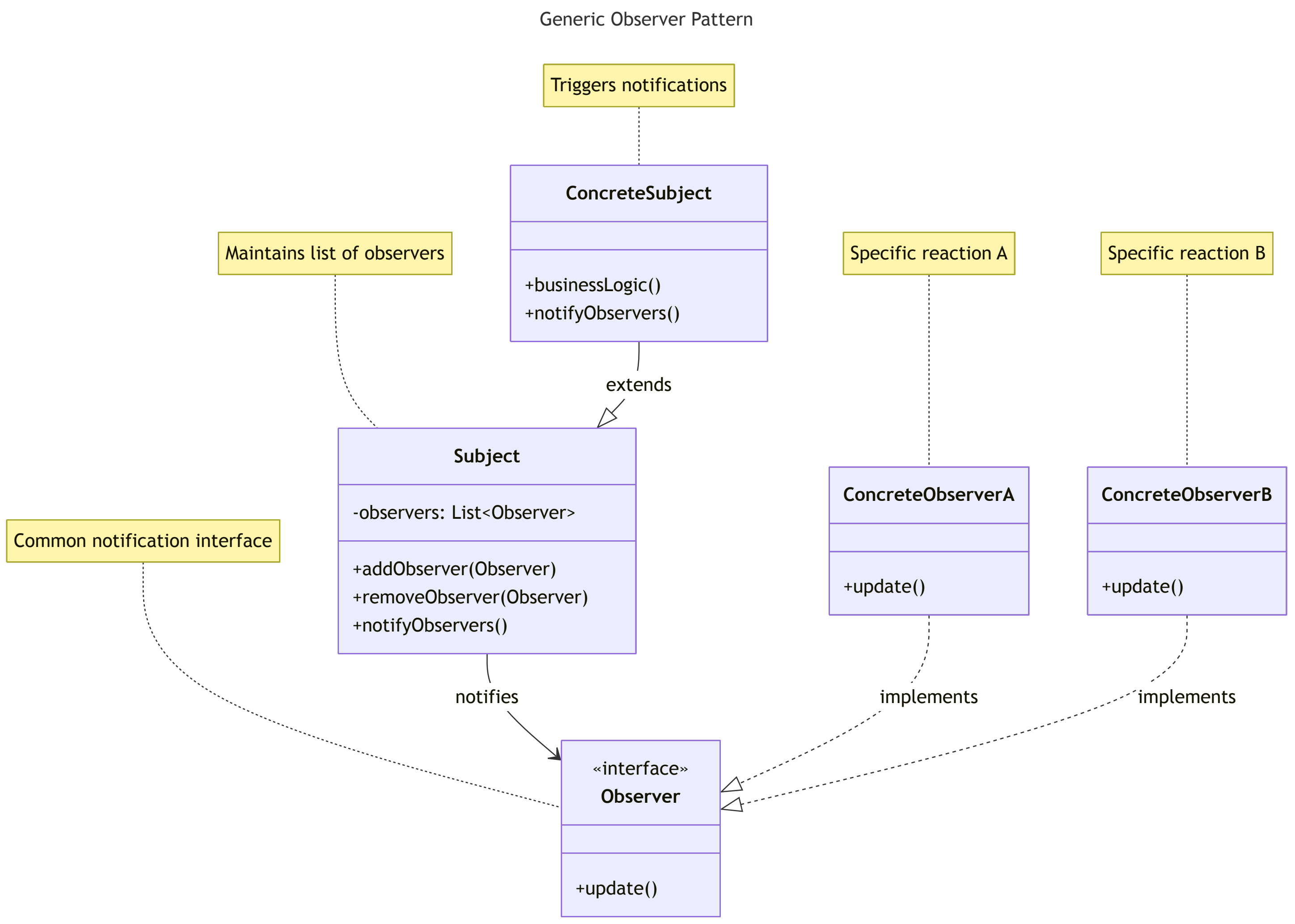
- TaskManager directly depends on concrete notification service classes
- High-level modules depending on low-level modules

🔴 **Interface Segregation Principle (ISP)**

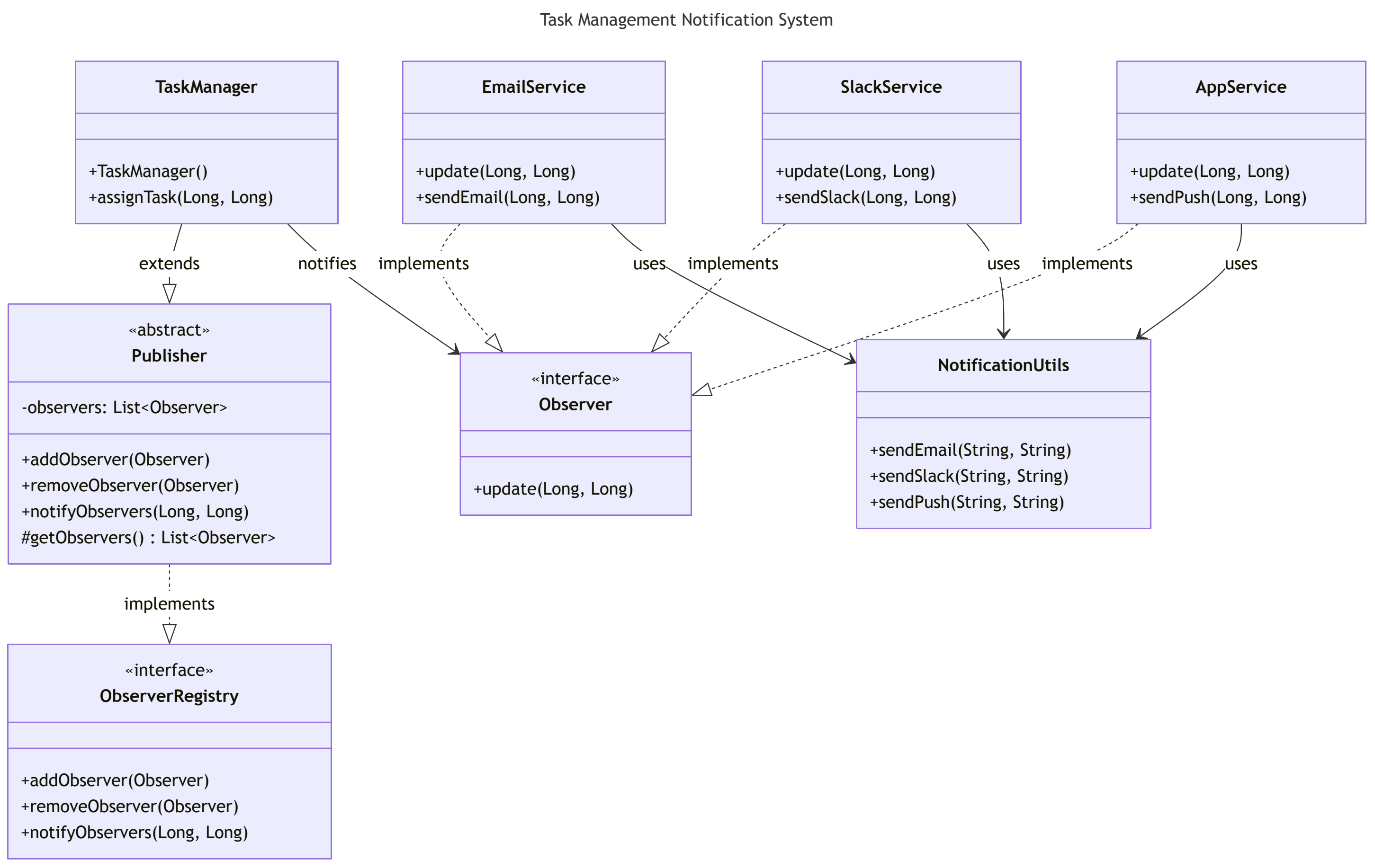
- No common interface for notification services
- Each service has different method signatures

With Observer Pattern

Generic Diagram



Specific Diagram



How Observer Pattern Helps Here

Key Benefits:

- **Loose Coupling**: TaskManager no longer directly depends on specific notification services
- **Dynamic Subscription**: Add/remove notification channels at runtime without stopping the system
- **Easy Extension**: Add new notification types (SMS, Teams, etc.) without modifying TaskManager
- **Single Responsibility**: TaskManager focuses on task assignment, observers handle notifications
- **Standardized Interface**: All notification services implement the same Observer interface
- **Runtime Flexibility**: Subscribe/unsubscribe observers based on user preferences or system configuration

Key Transformations

SOLID Principles Now Followed ✅

- **SRP**: TaskManager handles only task assignment, each observer handles one notification type
- **OCP**: Open for extension (new observers), closed for modification
- **DIP**: TaskManager depends on Observer abstraction, not concrete implementations
- **ISP**: Clean Observer interface with single responsibility

Usage Examples

```java
// Setup notification system ✅
TaskManager taskManager = new TaskManager();
EmailService emailService = new EmailService();
SlackService slackService = new SlackService();
AppService appService = new AppService();

// Subscribe observers ✅
taskManager.addObserver(emailService);
taskManager.addObserver(slackService);
taskManager.addObserver(appService);

// Assign task - automatically notifies all observers ✅
taskManager.assignTask(101L, 201L);

// Dynamic configuration - remove Slack for maintenance ✅
taskManager.removeObserver(slackService);
taskManager.assignTask(102L, 202L); // Only email and app notifications

// Add new notification type without changing TaskManager ✅
taskManager.addObserver(new SMSService()); // New observer!
taskManager.addObserver(new TeamsService()); // Another new observer!

// User preference-based notifications ✅
if (user.prefersEmail()) {
    taskManager.addObserver(emailService);
}
if (user.prefersSlack()) {
    taskManager.addObserver(slackService);
}
```