Семинар №12
Паттерны проектирования и GoF паттерны в Spring приложении

1. **Инструментарий:**

Урок
Презентация

---

2. **Цели семинара №12:**

- Понять основные концепции и принципы паттернов проектирования и GoF паттернов в Spring приложении по шаблону.
- Освоить практические навыки применения этих паттернов для повышения качества и поддерживаемости Spring приложений.

По итогам семинара №12 слушатель должен **знать**:

- Основные GoF паттерны (например, Singleton, Factory Method, Observer и т. д.) и их роли в разработке Spring приложений.
- Принципы и практические аспекты использования паттернов проектирования, в том числе их применение в Spring Framework.

По итогам семинара №12 слушатель должен **уметь**:

- Анализировать задачи и определять, какие паттерны проектирования могут быть применимы для их решения в контексте Spring приложений.
- Реализовывать и интегрировать выбранные паттерны в свои Spring проекты с целью повышения их качества, модульности и поддерживаемости.

---

3. **План Содержание:**

Этап урока	Тайминг, минуты	Формат
Введение, обзор темы	20	Модерирует преподаватель
Задание 1	40	Студенты выполняют, преподаватель

		помогает в решении проблем
Задание 2	40	Студенты выполняют, преподаватель помогает в решении проблем
Вопросы и обсуждение	20	Модерирует преподаватель
Длительность:	**120**	

4. Блок 1.

Тайминг:
Объяснение правил – 10 минут
Работа в команде – 30 минут

Задание:
1. Разработайте Spring приложение для управления инвентарем в компьютерном магазине. Примените паттерн фасад (Facade) для создания удобного интерфейса для клиентов магазина.
2. Реализуйте паттерн строитель (Builder) для создания заказов на комплектующие (например, процессоры, видеокарты и т. д.).
3. Используйте паттерн проектирования адаптер (Adapter) для интеграции сторонней системы проверки наличия комплектующих.

Пример решения с кодом:
1. Создайте класс `ComputerStoreFacade`, который предоставляет удобный интерфейс для клиентов магазина для управления инвентарем.

```
public class ComputerStoreFacade {
    private InventorySystem inventorySystem;
    private OrderBuilder orderBuilder;

    public ComputerStoreFacade() {
        inventorySystem = new InventorySystem();
        orderBuilder = new OrderBuilder();
    }

    public void displayAvailableComponents() {
        inventorySystem.displayAvailableComponents();
    }

    public void createCustomOrder(String customerName) {
```

```
        orderBuilder.createOrder(customerName);
    }

    public void addComponentToOrder(String componentName) {
        orderBuilder.addComponent(componentName);
    }

    public Order finalizeOrder() {
        return orderBuilder.build();
    }
}
```

2. Реализуйте паттерн Builder для создания заказов на комплектующие. Создайте классы для комплектующих и заказов.

```
public class Component {
    // Поля и методы компонента
}

public class Order {
    private String customerName;
    private List<Component> components;

    // Методы для добавления компонентов и других операций с заказом
}

public class OrderBuilder {
    private Order order;

    public void createOrder(String customerName) {
        order = new Order();
        order.setCustomerName(customerName);
    }

    public void addComponent(String componentName) {
        Component component = new Component(componentName);
        order.addComponent(component);
    }

    public Order build() {
        return order;
```

```
    }
}
```

3. Используйте паттерн Adapter для интеграции сторонней системы проверки наличия комплектующих в магазине.

```
public interface InventoryChecker {
    boolean isComponentAvailable(String componentName);
}

public class ThirdPartyInventoryCheckerAdapter implements
InventoryChecker {
    private ThirdPartyInventorySystem thirdPartySystem;

    public ThirdPartyInventoryCheckerAdapter(ThirdPartyInventorySystem
thirdPartySystem) {
        this.thirdPartySystem = thirdPartySystem;
    }

    @Override
    public boolean isComponentAvailable(String componentName) {
        // Адаптация вызова к сторонней системе
        return thirdPartySystem.checkComponentAvailability(componentName);
    }
}
```

Часто встречающиеся ошибки:
1. Неясная реализация паттернов, что может привести к неправильному использованию.
2. Отсутствие обработки исключений и проверок на некорректные данные.
3. Недостаточная документация и комментарии к коду.
4. Неудачная адаптация сторонней системы с использованием паттерна Adapter.

5. Блок 2.

Тайминг:
Объяснение правил – 10 минут
Работа в команде – 30 минут

Задание:
1. Создайте Spring приложение для управления задачами (To-Do List). Примените паттерн Singleton для создания сервиса управления задачами.
2. Реализуйте паттерн Observer для отслеживания изменений в списке задач и автоматического оповещения пользователей о добавлении или удалении задач.
3. Используйте паттерн Factory Method для создания различных типов задач (например, задачи срочного и обычного выполнения).

Пример решения с кодом:
1. Создайте класс `TaskManager` и примените паттерн Singleton к этому классу. Этот класс будет отвечать за управление задачами.

```
public class TaskManager {
   private static TaskManager instance = new TaskManager();

   private TaskManager() {
      // Приватный конструктор для Singleton
   }

   public static TaskManager getInstance() {
      return instance;
   }

   // Другие методы управления задачами
}
```

2. Создайте интерфейс `TaskObserver` и реализуйте его в классах, которые хотят получать уведомления о изменениях в списке задач.

```
public interface TaskObserver {
   void update(Task task);
}
```

3. Реализуйте фабричный метод для создания разных типов задач. Создайте интерфейс `TaskFactory` и конкретные реализации этого интерфейса для создания задач разных типов.

```
public interface TaskFactory {
   Task createTask();
}

public class UrgentTaskFactory implements TaskFactory {
   @Override
   public Task createTask() {
      return new UrgentTask();
   }
}

public class RegularTaskFactory implements TaskFactory {
   @Override
   public Task createTask() {
      return new RegularTask();
   }
}
```

Часто встречающиеся ошибки:
1. Неправильная реализация Singleton, например, создание нескольких экземпляров `TaskManager`.
2. Неясное разделение обязанностей между классами и неправильное применение фабричного метода.
3. Отсутствие обработки исключений и проверки на неверные входные данные.
4. Недостаточная документация и комментарии к коду.
5. Отсутствие интерфейсов и абстракций, что затрудняет расширение функциональности приложения.

6. Домашнее задание

Задание:
1. Создайте Spring приложение для управления задачами (Task Management). Примените паттерн Singleton для создания сервиса управления задачами.
2. Реализуйте паттерн Observer для отслеживания изменений в состоянии задач и оповещения об этих изменениях подписчиков.
3. Используйте паттерн фабрики (Factory Method) для создания разных типов задач (например, задачи срочного и обычного выполнения).

Пример решения с кодом:
1. Создайте класс `TaskManager` и примените паттерн Singleton к этому классу. Этот класс будет отвечать за управление задачами.

```
public class TaskManager {
   private static TaskManager instance = new TaskManager();

   private TaskManager() {
      // Приватный конструктор для Singleton
   }

   public static TaskManager getInstance() {
      return instance;
   }

   // Другие методы управления задачами
}
```

2. Создайте интерфейс `TaskObserver` и реализуйте его в классах, которые хотят получать уведомления о изменениях в задачах.

```
public interface TaskObserver {
   void update(Task task);
}
```

3. Реализуйте фабричный метод для создания разных типов задач. Создайте интерфейс `TaskFactory` и конкретные реализации этого интерфейса для создания задач разных типов.

```
public interface TaskFactory {
   Task createTask();
}

public class UrgentTaskFactory implements TaskFactory {
   @Override
   public Task createTask() {
      return new UrgentTask();
   }
}

public class RegularTaskFactory implements TaskFactory {
   @Override
   public Task createTask() {
      return new RegularTask();
   }
}
```

Рекомендации для преподавателей по оценке задания:

1. Оцените правильность применения паттернов проектирования в решении задачи.
2. Проверьте, что код решения соответствует заданию и что он логично структурирован.
3. Оцените наличие документации и комментариев к коду, которые делают код более понятным для других разработчиков.
4. Удостоверьтесь, что нет явных ошибок или исключений в коде.
5. Оцените качество реализации и правильность использования языка программирования и Spring Framework.