Sri Lanka Institute of Information Technology

SE2030-Software Engineering

2025-Y2-S1-KU-34

Web-Based Catering System

Design Pattern

Group Members :

Reg No	**Name**
IT24610823	Shaahidh M.W.M
IT24104096	Kaluarachchi S.S
IT24102659	Senevirathna K.S.D.B
IT24103086	Wanasundara W.A.A.L
IT24103607	Lakshani K.A.A
IT24102908	Galapitage S.A

Design Pattern(s) Used :

- Singleton Pattern
- Factory Pattern

01. Singleton Pattern

Where Used : All @Service, @Repository, @Controller, and @Component annotated classes

Files: all Services, all repositories, all controllers, DataInitializer.

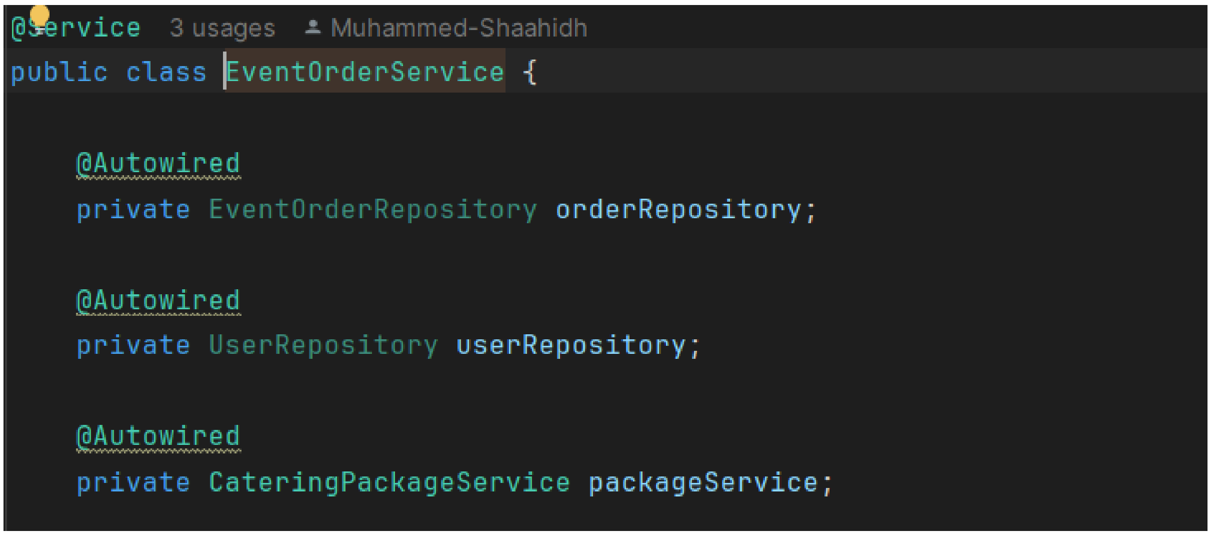
Justification:

Why This Pattern Was Chosen:

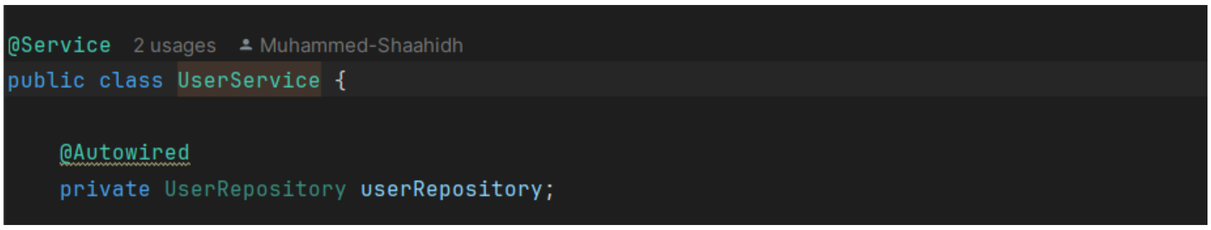
1. **Resource Efficiency**
    - Creating multiple instances of service/repository classes wastes memory
    - Services are stateless - no need for multiple copies
    - Single instance serves all requests efficiently
2. **Consistency**
    - All parts of application use same service instance
    - Ensures consistent business logic execution
    - Prevents conflicting state across multiple instances
3. **Centralized Control**
    - Single point of access for business operations
    - Easier to manage dependencies and lifecycle
    - Simplifies testing and debugging
4. **Spring Framework Best Practice**
    - Default scope for Spring beans

- Thread-safe by design
- Managed automatically by Spring container

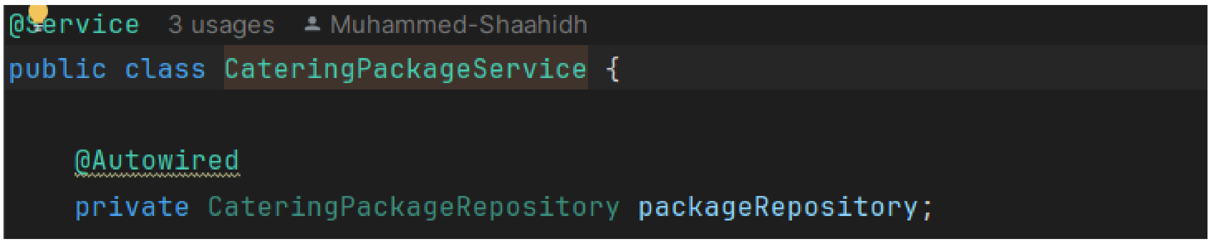
- Service Layer



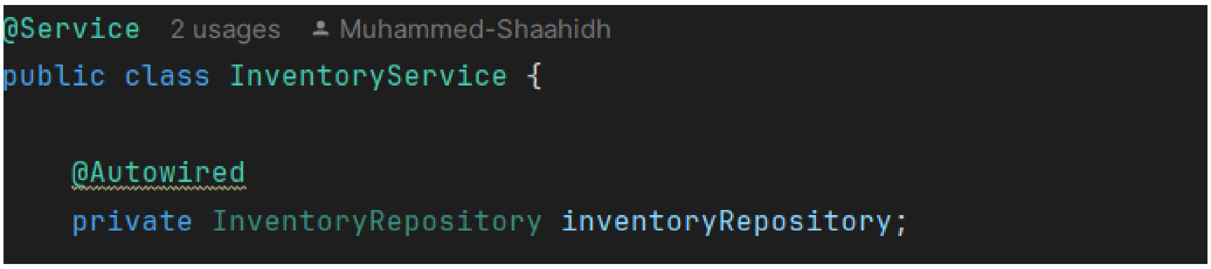
```
@Service  3 usages  Muhammed-Shaahidh
public class EventOrderService {

    @Autowired
    private EventOrderRepository orderRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private CateringPackageService packageService;
```



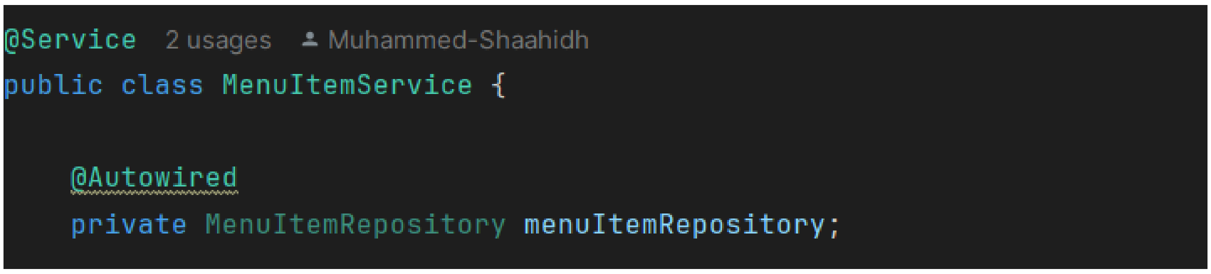
```
@Service  2 usages  Muhammed-Shaahidh
public class UserService {

    @Autowired
    private UserRepository userRepository;
```



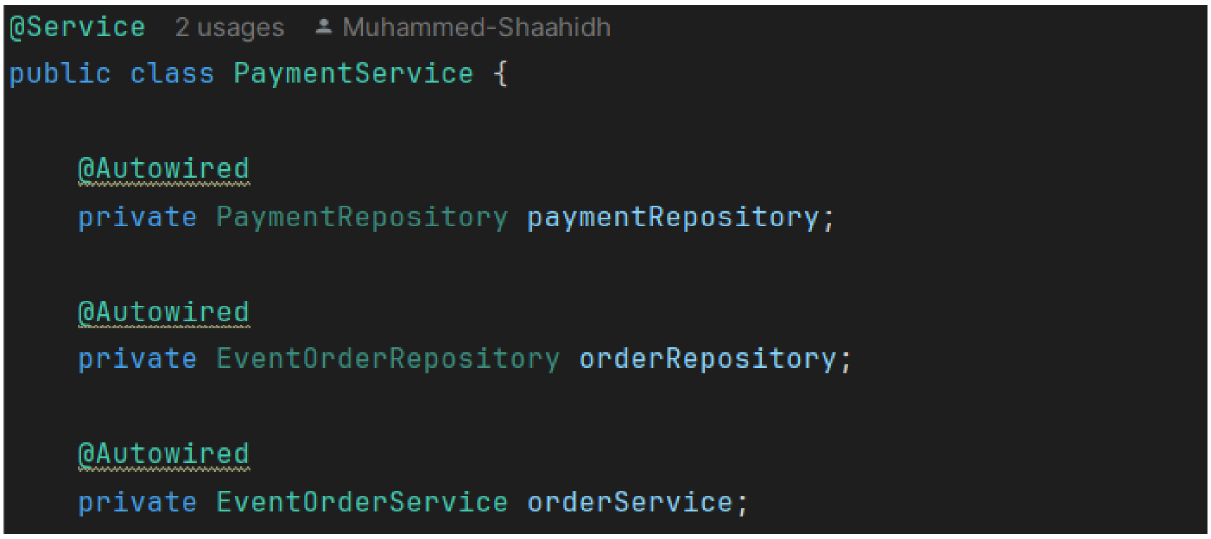
```
@Service  3 usages  Muhammed-Shaahidh
public class CateringPackageService {

    @Autowired
    private CateringPackageRepository packageRepository;
```



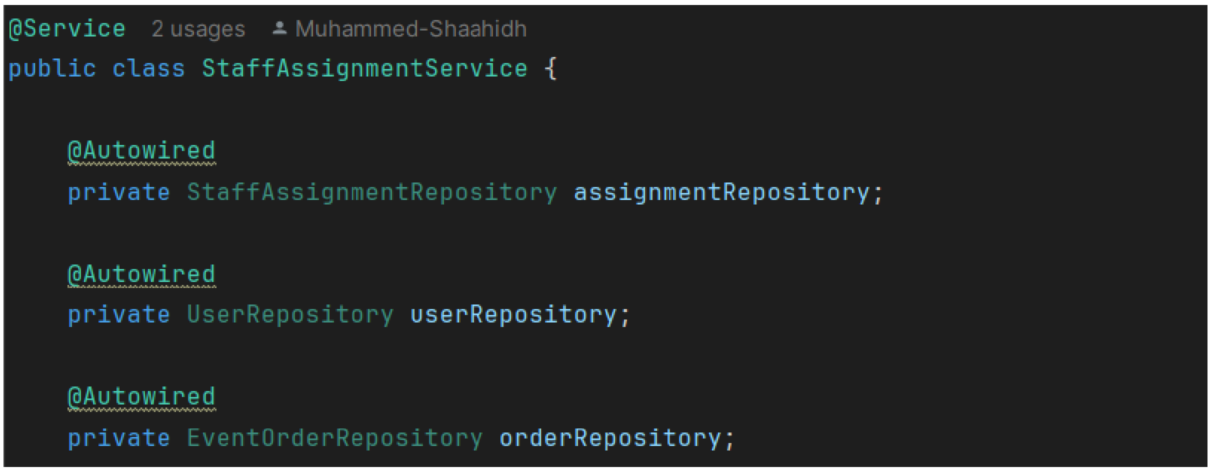
```
@Service  2 usages  Muhammed-Shaahidh
public class InventoryService {

    @Autowired
    private InventoryRepository inventoryRepository;
```



```java
@Service  2 usages  Muhammed-Shaahidh
public class MenuItemService {

    @Autowired
    private MenuItemRepository menuItemRepository;
```

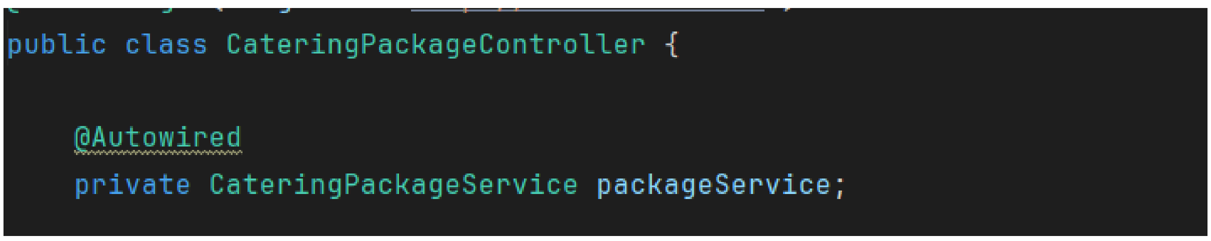


```java
@Service  2 usages  Muhammed-Shaahidh
public class PaymentService {

    @Autowired
    private PaymentRepository paymentRepository;

    @Autowired
    private EventOrderRepository orderRepository;

    @Autowired
    private EventOrderService orderService;
```

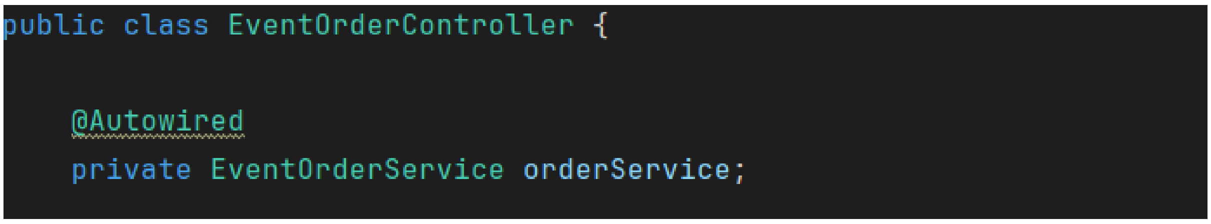


```java
@Service  2 usages  Muhammed-Shaahidh
public class StaffAssignmentService {

    @Autowired
    private StaffAssignmentRepository assignmentRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private EventOrderRepository orderRepository;
```

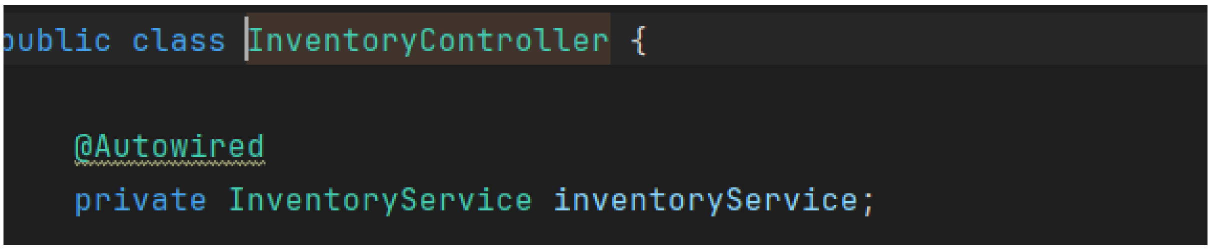
- Controller Layer



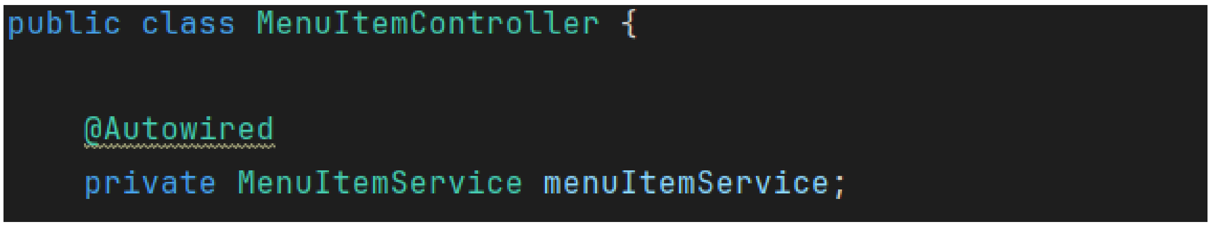
```java
public class CateringPackageController {

    @Autowired
    private CateringPackageService packageService;
```



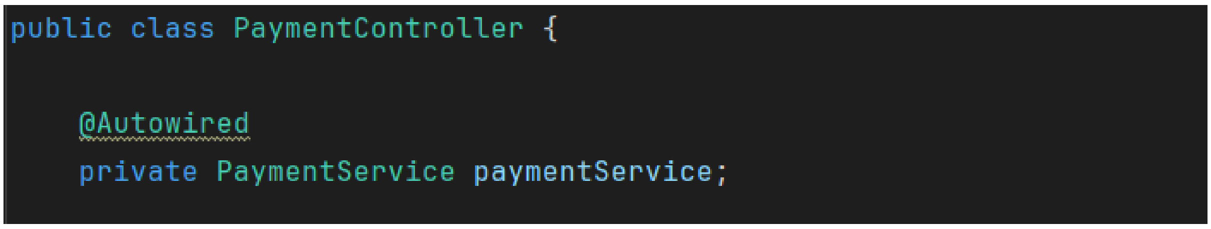
```java
public class EventOrderController {

    @Autowired
    private EventOrderService orderService;
```



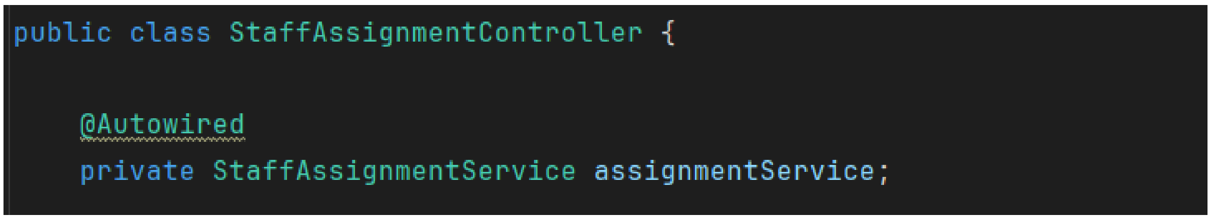
```java
public class InventoryController {

    @Autowired
    private InventoryService inventoryService;
```



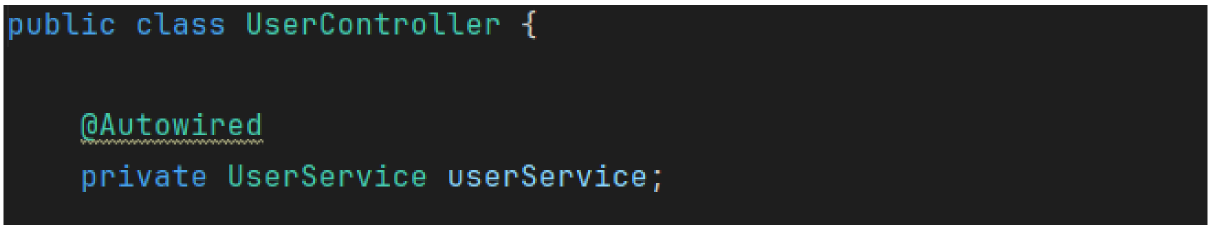
```java
public class MenuItemController {

    @Autowired
    private MenuItemService menuItemService;
```



```java
public class PaymentController {

    @Autowired
    private PaymentService paymentService;
```

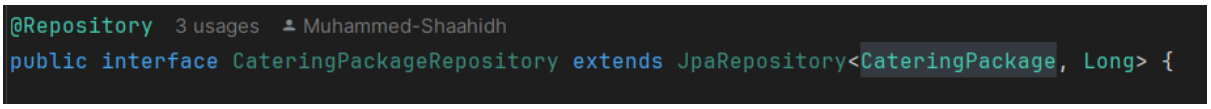


```java
public class StaffAssignmentController {

    @Autowired
    private StaffAssignmentService assignmentService;
```

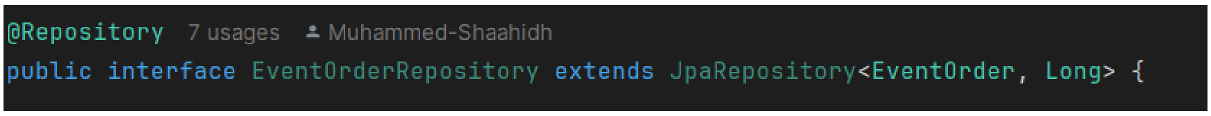


```java
public class UserController {

    @Autowired
    private UserService userService;
```

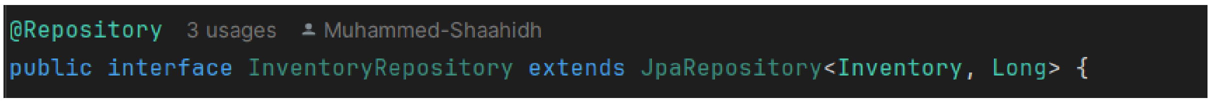
- Repository Layer



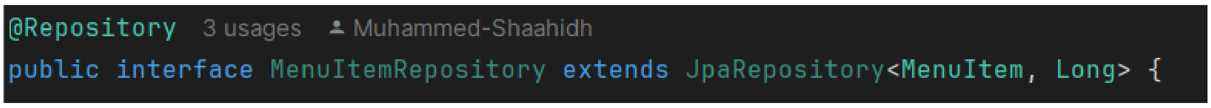
```java
@Repository  3 usages  Muhammed-Shaahidh
public interface CateringPackageRepository extends JpaRepository<CateringPackage, Long> {
```



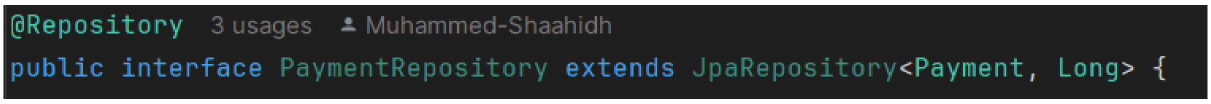
```java
@Repository  7 usages  Muhammed-Shaahidh
public interface EventOrderRepository extends JpaRepository<EventOrder, Long> {
```



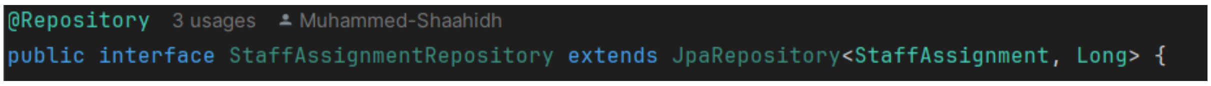
```
@Repository  3 usages  Muhammed-Shaahidh
public interface InventoryRepository extends JpaRepository<Inventory, Long> {
```



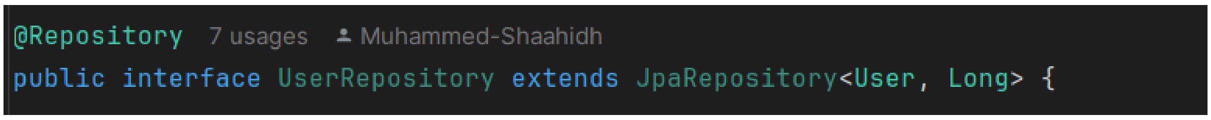
```
@Repository  3 usages  Muhammed-Shaahidh
public interface MenuItemRepository extends JpaRepository<MenuItem, Long> {
```



```
@Repository  3 usages  Muhammed-Shaahidh
public interface PaymentRepository extends JpaRepository<Payment, Long> {
```

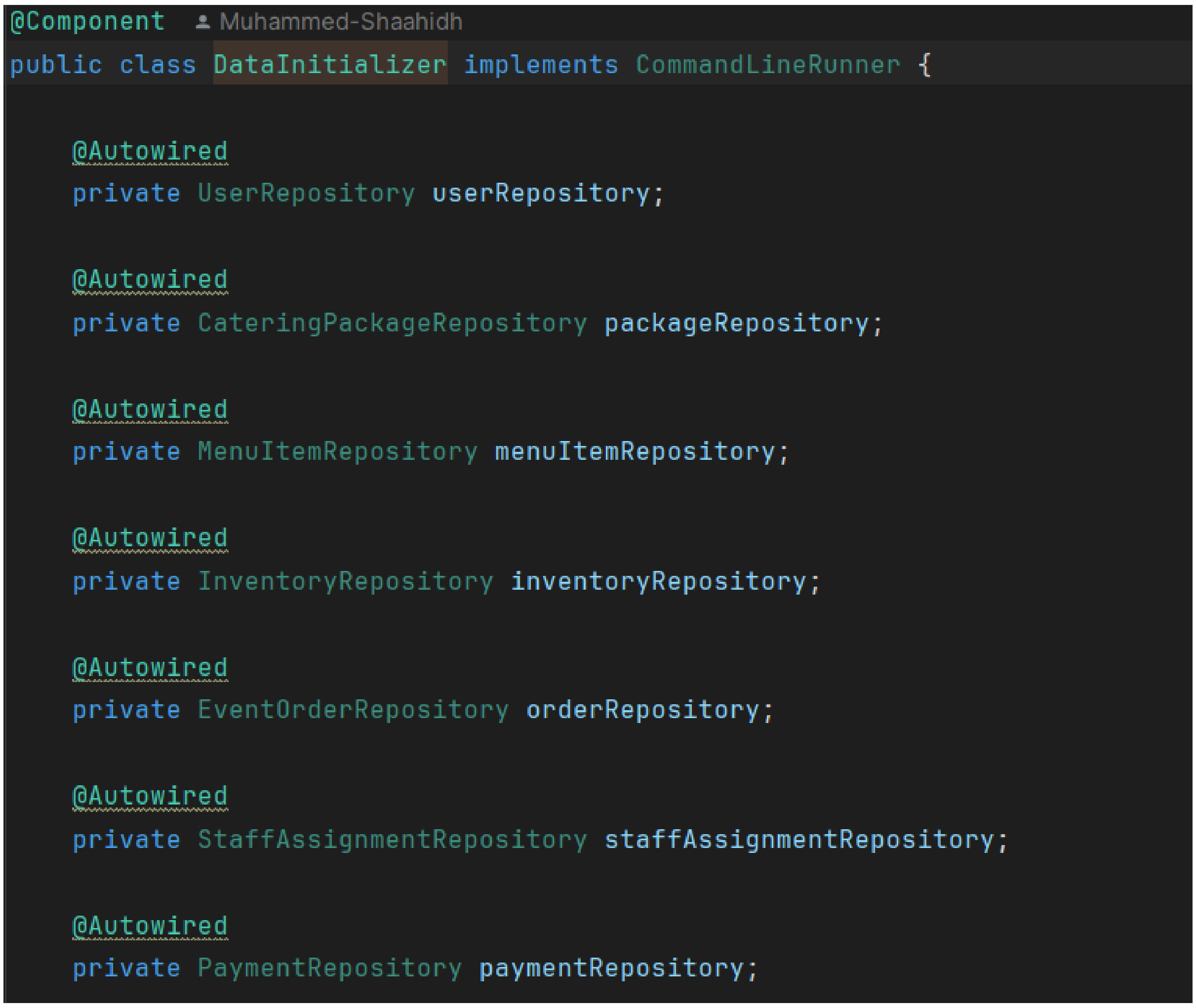


```
@Repository  3 usages  Muhammed-Shaahidh
public interface StaffAssignmentRepository extends JpaRepository<StaffAssignment, Long> {
```



```
@Repository  7 usages  Muhammed-Shaahidh
public interface UserRepository extends JpaRepository<User, Long> {
```

➢ Component



```
@Component  Muhammed-Shaahidh
public class DataInitializer implements CommandLineRunner {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private CateringPackageRepository packageRepository;

    @Autowired
    private MenuItemRepository menuItemRepository;

    @Autowired
    private InventoryRepository inventoryRepository;

    @Autowired
    private EventOrderRepository orderRepository;

    @Autowired
    private StaffAssignmentRepository staffAssignmentRepository;

    @Autowired
    private PaymentRepository paymentRepository;
```

02. Factory Pattern

Where Used: DataInitializer.java (lines 95-303)

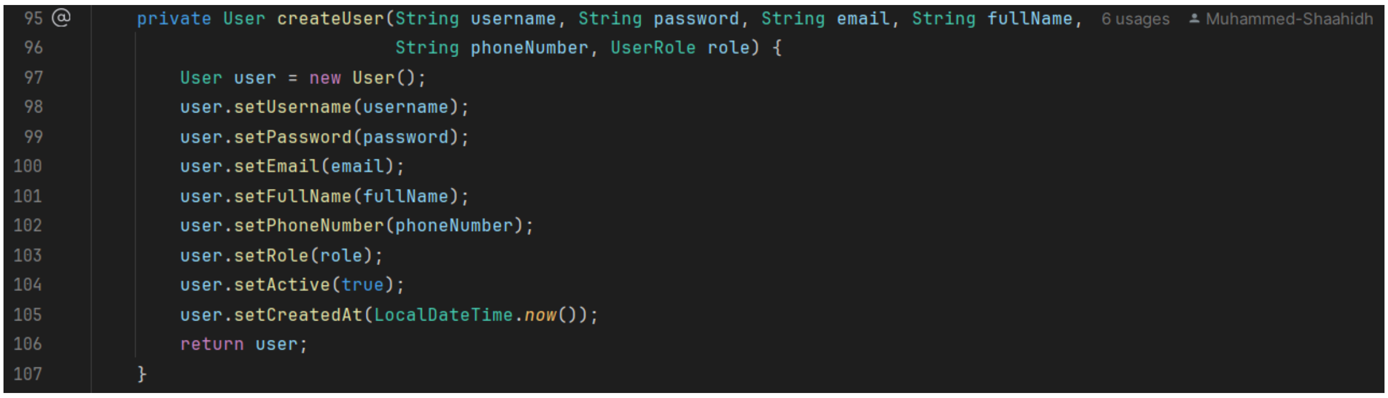
Factory Methods: createUser(), createMenuItem(), createPackage(), createOrder(), createPayment(), createAssignment(), createInventoryItem()

Justification:

Why This Pattern Was Chosen:

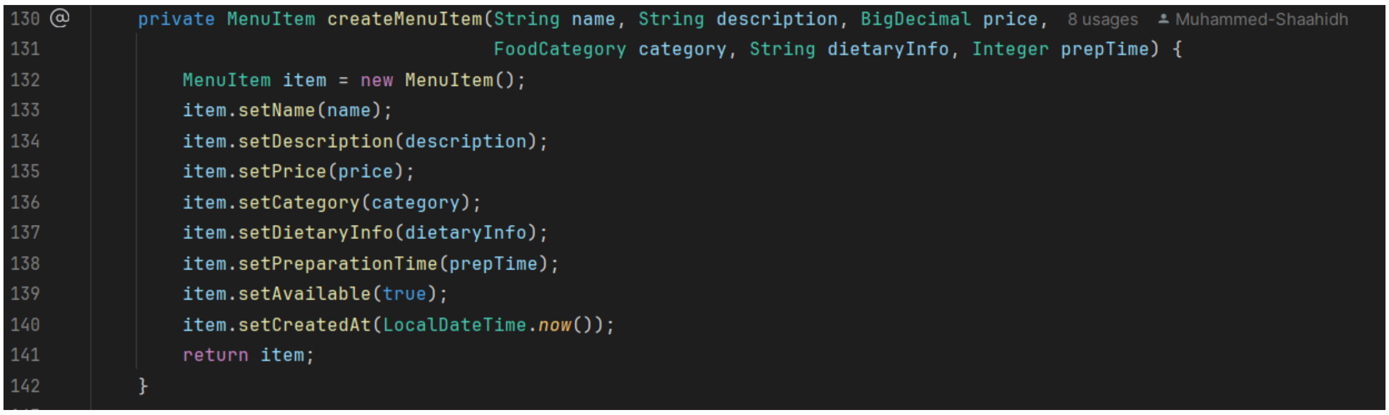
1. **Complex Object Creation**
    - Entities like User, EventOrder, Payment have multiple fields
    - Need default values, timestamps, status initialization
    - Example: EventOrder requires calculating totalAmount, setting createdAt, initializing status and paymentStatus
2. **Consistency in Object Construction**
    - Without factory: Risk of forgetting to set required fields
    - With factory: Guaranteed complete and valid objects every time
    - Example from line 228: order.setTotalAmount(pkg.getPrice().multiply(BigDecimal.valueOf(guests)))
3. **Code Reusability**
    - Lines 78-92: Creates 6 different users using same factory method
    - Reduces code duplication from ~10 lines per user to 1 line
4. **Maintainability**
    - If User entity changes (e.g., add new required field), only modify factory method
    - Don't need to update 50+ places where User objects are created

➢ User Factory



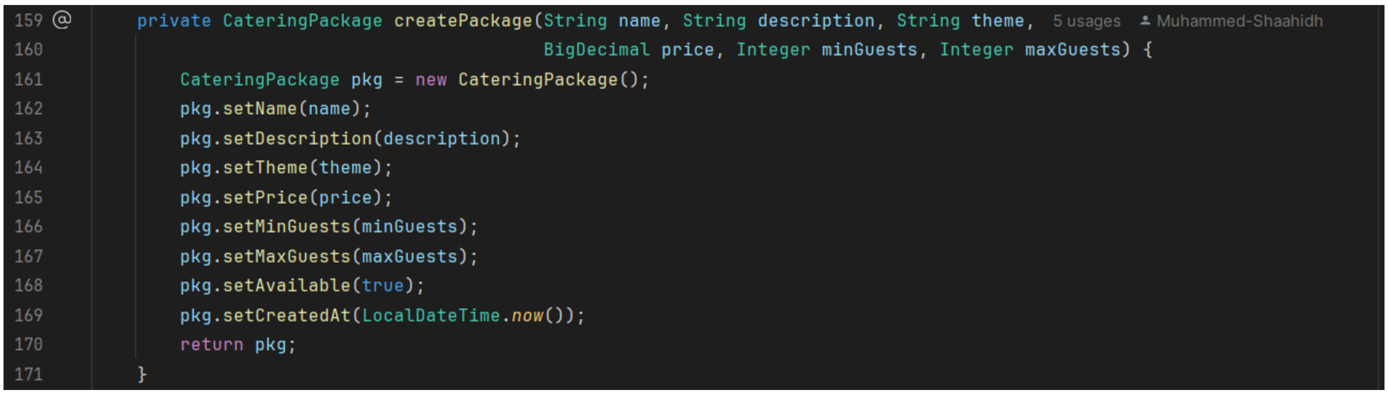
```java
private User createUser(String username, String password, String email, String fullName,  6 usages  Muhammed-Shaahidh
                        String phoneNumber, UserRole role) {
    User user = new User();
    user.setUsername(username);
    user.setPassword(password);
    user.setEmail(email);
    user.setFullName(fullName);
    user.setPhoneNumber(phoneNumber);
    user.setRole(role);
    user.setActive(true);
    user.setCreatedAt(LocalDateTime.now());
    return user;
}
```

➢ MenuItem Factory



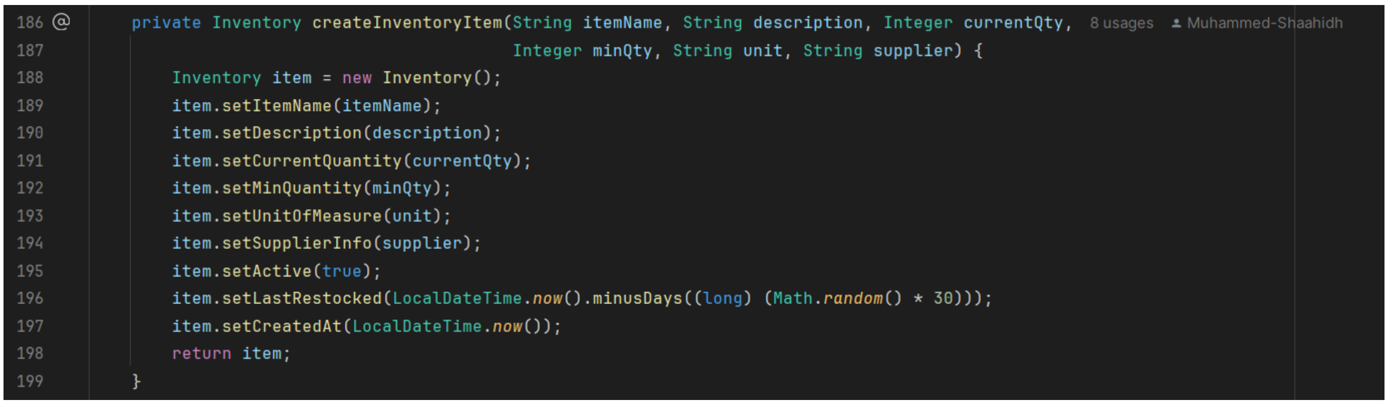
```java
private MenuItem createMenuItem(String name, String description, BigDecimal price,  8 usages  Muhammed-Shaahidh
                                FoodCategory category, String dietaryInfo, Integer prepTime) {
    MenuItem item = new MenuItem();
    item.setName(name);
    item.setDescription(description);
    item.setPrice(price);
    item.setCategory(category);
    item.setDietaryInfo(dietaryInfo);
    item.setPreparationTime(prepTime);
    item.setAvailable(true);
    item.setCreatedAt(LocalDateTime.now());
    return item;
}
```

➢ CetringPackage Factory



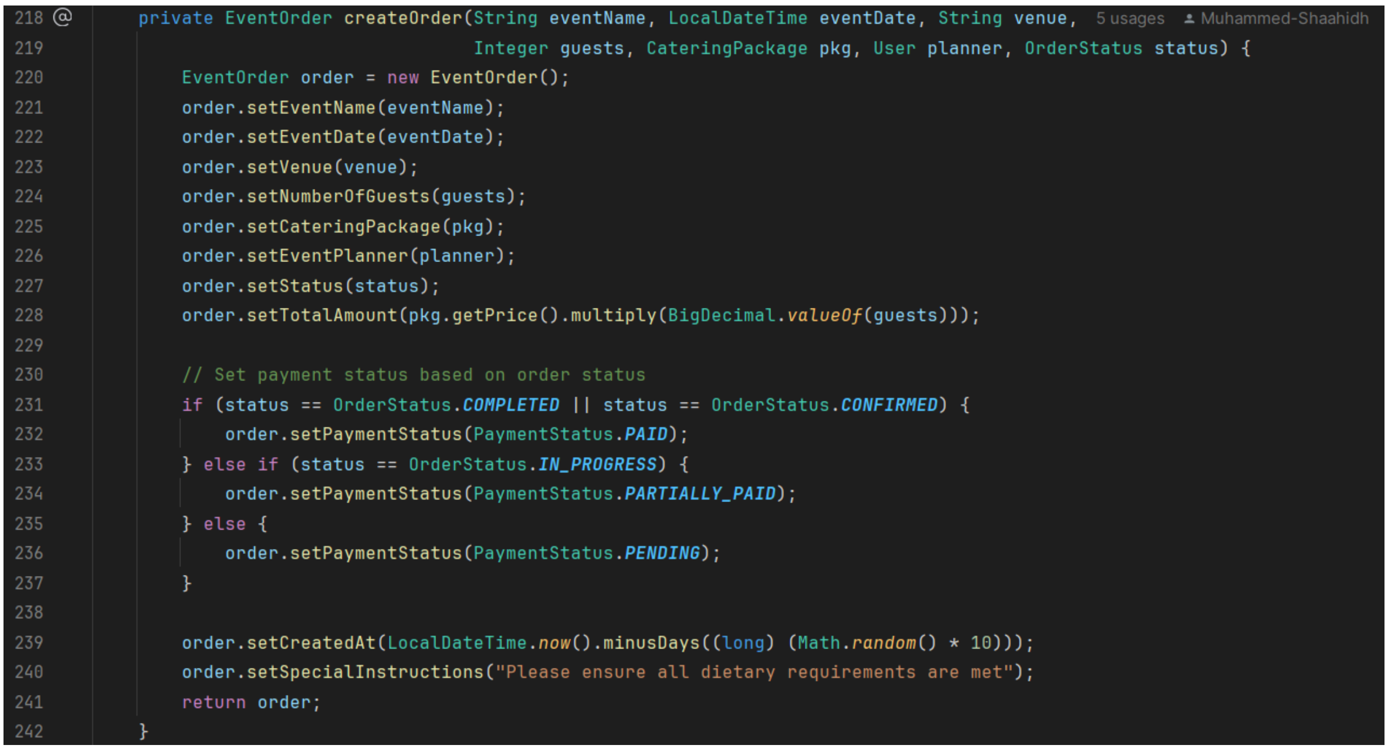
```java
private CateringPackage createPackage(String name, String description, String theme,  5 usages  Muhammed-Shaahidh
                                      BigDecimal price, Integer minGuests, Integer maxGuests) {
    CateringPackage pkg = new CateringPackage();
    pkg.setName(name);
    pkg.setDescription(description);
    pkg.setTheme(theme);
    pkg.setPrice(price);
    pkg.setMinGuests(minGuests);
    pkg.setMaxGuests(maxGuests);
    pkg.setAvailable(true);
    pkg.setCreatedAt(LocalDateTime.now());
    return pkg;
}
```

➢ Inventory Factory



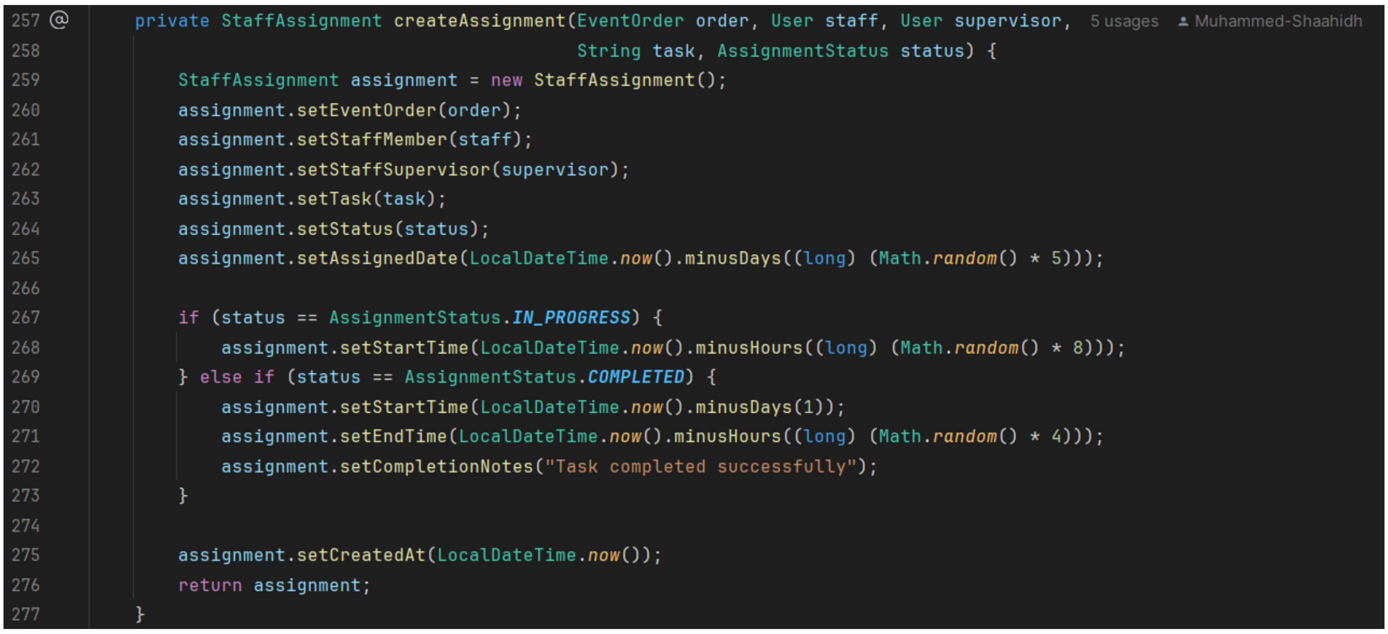
```
    private Inventory createInventoryItem(String itemName, String description, Integer currentQty,  8 usages  Muhammed-Shaahidh
                                          Integer minQty, String unit, String supplier) {
        Inventory item = new Inventory();
        item.setItemName(itemName);
        item.setDescription(description);
        item.setCurrentQuantity(currentQty);
        item.setMinQuantity(minQty);
        item.setUnitOfMeasure(unit);
        item.setSupplierInfo(supplier);
        item.setActive(true);
        item.setLastRestocked(LocalDateTime.now().minusDays((long) (Math.random() * 30)));
        item.setCreatedAt(LocalDateTime.now());
        return item;
    }
```

➢ EventOrder Factory



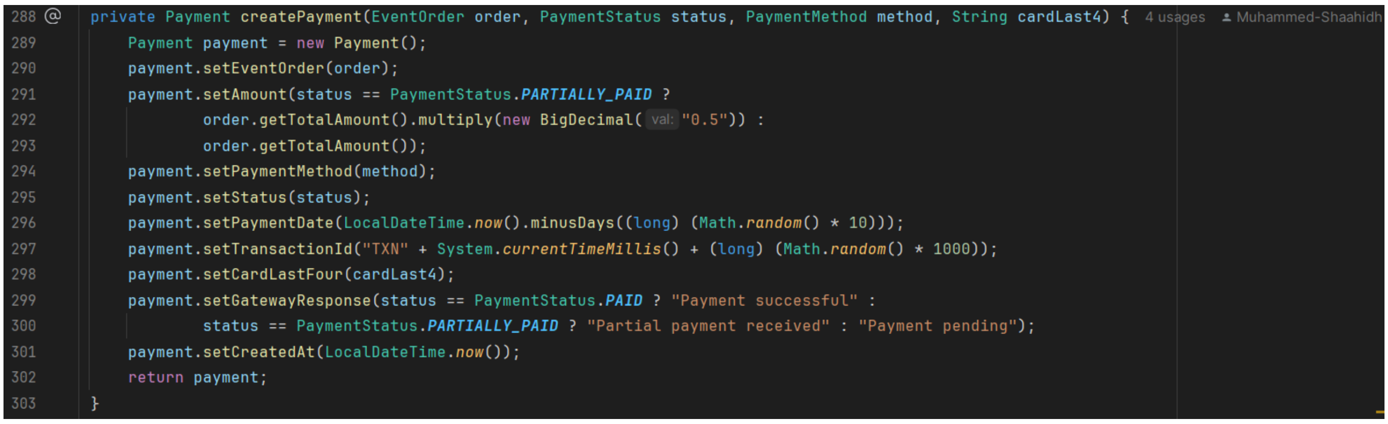
```
    private EventOrder createOrder(String eventName, LocalDateTime eventDate, String venue,  5 usages  Muhammed-Shaahidh
                                   Integer guests, CateringPackage pkg, User planner, OrderStatus status) {
        EventOrder order = new EventOrder();
        order.setEventName(eventName);
        order.setEventDate(eventDate);
        order.setVenue(venue);
        order.setNumberOfGuests(guests);
        order.setCateringPackage(pkg);
        order.setEventPlanner(planner);
        order.setStatus(status);
        order.setTotalAmount(pkg.getPrice().multiply(BigDecimal.valueOf(guests)));

        // Set payment status based on order status
        if (status == OrderStatus.COMPLETED || status == OrderStatus.CONFIRMED) {
            order.setPaymentStatus(PaymentStatus.PAID);
        } else if (status == OrderStatus.IN_PROGRESS) {
            order.setPaymentStatus(PaymentStatus.PARTIALLY_PAID);
        } else {
            order.setPaymentStatus(PaymentStatus.PENDING);
        }

        order.setCreatedAt(LocalDateTime.now().minusDays((long) (Math.random() * 10)));
        order.setSpecialInstructions("Please ensure all dietary requirements are met");
        return order;
    }
```

- StaffAssignment Factory



```java
    private StaffAssignment createAssignment(EventOrder order, User staff, User supervisor,  5 usages  Muhammed-Shaahidh
                                             String task, AssignmentStatus status) {
        StaffAssignment assignment = new StaffAssignment();
        assignment.setEventOrder(order);
        assignment.setStaffMember(staff);
        assignment.setStaffSupervisor(supervisor);
        assignment.setTask(task);
        assignment.setStatus(status);
        assignment.setAssignedDate(LocalDateTime.now().minusDays((long) (Math.random() * 5)));

        if (status == AssignmentStatus.IN_PROGRESS) {
            assignment.setStartTime(LocalDateTime.now().minusHours((long) (Math.random() * 8)));
        } else if (status == AssignmentStatus.COMPLETED) {
            assignment.setStartTime(LocalDateTime.now().minusDays(1));
            assignment.setEndTime(LocalDateTime.now().minusHours((long) (Math.random() * 4)));
            assignment.setCompletionNotes("Task completed successfully");
        }

        assignment.setCreatedAt(LocalDateTime.now());
        return assignment;
    }
```

- Payment Factory

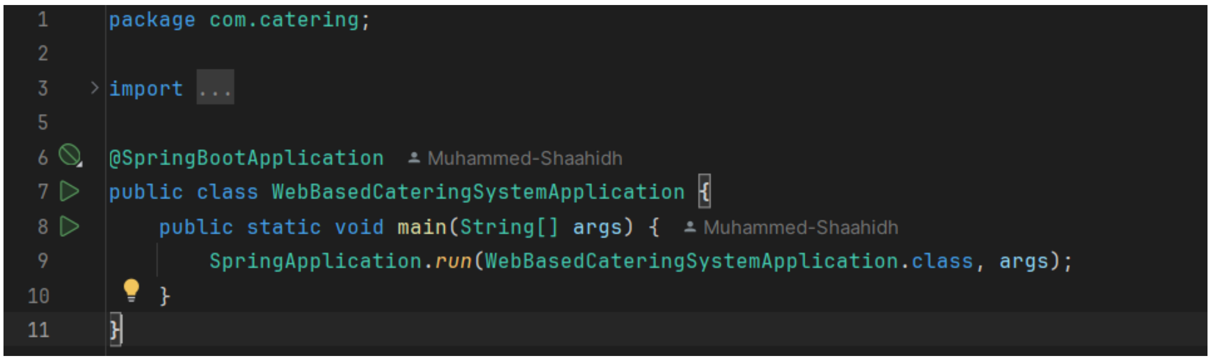


```java
private Payment createPayment(EventOrder order, PaymentStatus status, PaymentMethod method, String cardLast4) {  4 usages  Muhammed-Shaahidh
    Payment payment = new Payment();
    payment.setEventOrder(order);
    payment.setAmount(status == PaymentStatus.PARTIALLY_PAID ?
            order.getTotalAmount().multiply(new BigDecimal( val: "0.5")) :
            order.getTotalAmount());
    payment.setPaymentMethod(method);
    payment.setStatus(status);
    payment.setPaymentDate(LocalDateTime.now().minusDays((long) (Math.random() * 10)));
    payment.setTransactionId("TXN" + System.currentTimeMillis() + (long) (Math.random() * 1000));
    payment.setCardLastFour(cardLast4);
    payment.setGatewayResponse(status == PaymentStatus.PAID ? "Payment successful" :
            status == PaymentStatus.PARTIALLY_PAID ? "Partial payment received" : "Payment pending");
    payment.setCreatedAt(LocalDateTime.now());
    return payment;
}
```

Proper use of Java classes

Main Class – (WebBasedCateringSystemApplication)

- **Purpose** :- Entry point for the Spring Boot application



```java
package com.catering;

import ...

@SpringBootApplication  Muhammed-Shaahidh
public class WebBasedCateringSystemApplication {
    public static void main(String[] args) {  Muhammed-Shaahidh
        SpringApplication.run(WebBasedCateringSystemApplication.class, args);
    }
}
```

Clear naming conventions and comments for better readability

1. **Classes:** PascalCase
   - UserService, EventOrderController, CateringPackage
2. **Methods:** camelCase
   - createUser(), getAllOrders(), updatePaymentStatus()
3. **Variables: camelCase**
   - username, eventDate, totalAmount
4. **Constants/Enums:** UPPER_SNAKE_CASE
   - BUSINESS_OWNER, EVENT_PLANNER, PENDING, APPROVED
5. **Packages:** lowercase
   - com.catering.entity, com.catering.service, com.catering.controller