Code Smell Write-Up - Team 32

Team Members: Arish Virani, Rafay Hadi, Muneeb Ali Asif, Saibalaji Nagarajan, Rayan Castilla Zouine, Mitchell Richard Ford

a) Identifying Code Smells:

1. **Bloater: Long Method:** The calculateTotalPrice() method in the Order class has multiple responsibilities like calculating total price, applying discounts, and checking for gift cards, making it too complex and long.
2. **Dispensable: Duplicate Code:** The logic to send confirmation emails is in both Order and EmailSender classes, which is unnecessary.
3. **Bloater: Large Class:** The Order class has too many methods and code, including managing item details, customer information, and email notification, which is against the Single Responsibility Principle and cannot be effectively handled.
4. **Object-Orientation Abusers: Switch Statements:** The Order class has multiple switch statements for different discount types and calculation strategies. This can lead to issues in scalability later, when adding new discount types as the switch cases would have to be altered directly.
5. **Object-Orientation Abusers: Refused Bequest:** The DiscountType enum is too rigid and could be better represented using the Strategy Pattern, for more flexibility and easier maintenance when addition of new discount types is needed.
6. **Dispensable: Data Class:** The properties in the Item class are directly accessed via public getter methods, which could lead to unintended modifications and over exposure.
7. **Poor Naming Conventions:** The class name main does not follow conventions and hasGiftCard() method is slightly unclear.

b) Fixing Code Smells:

1. **Refactor Long Method: Extract Method**: Broke down the calculateTotalPrice() method in the Order class into smaller and focused methods like applyGiftCardDiscount() and applyOrderDiscount() to make more readable.
2. **Eliminate Duplicate Code: Extract Class**: Introduced a dedicated class OrderNotification that handles all email notifications. This separates email logic from order management, following the Single Responsibility Principle.
3. **Refactor Large Class: Extract Class**: Moved email notification logic out of the Order class and into OrderNotification, the responsibilities of the Order class are reduced.
4. **Eliminate Switch Statements: Decompose Conditionals + Extract Classes:** Replaced the multiple switch statements in the Order class with a strategy pattern implementation. Defined a Discount interface with a method for calculating the discount, and created concrete classes for each discount type, PercentageDiscount and AmountDiscount. This approach enhances scalability by

allowing new discount types to be added as new classes without modifying existing code, thereby following the Open/Closed Principle.

5. **Implement Strategy Pattern**: Replaced the DiscountType enum with a Discount interface and concrete classes PercentageDiscount and AmountDiscount to allow for dynamic discount calculations.
6. **Enhance Encapsulation:** Modified the Item class to have a getTotalPrice() method that encapsulates the price calculation, to prevent over exposure of specific variables.
7. **Renaming:** Rename the main class to Main to follow Java naming conventions. Also, renamed hasGiftCard() to containsGiftCard() to make method more clear.