

# Abdulhaq Zulfiqar

22i-2585

SE-G

SQE A3

# Contents

Java Code Review Checklist for EmployeeManagement	6
Refactoring Repeated File Handling Code	12
Refactoring Username Generation in add Method	12
Refactoring Hardcoded Values and Logging	12
Appendix:	13
2. Java Code Review Checklist for Inventory	16
Refactoring Tasks	22
Refactoring Task 1: Separate File Reading and Writing Logic	22
Refactoring Task 2: Cache Item Lookup to Improve Performance	22
Refactoring Task 3: Replace System.out.println with Logging	22
Appendix:	22
3. Java Code Review Checklist for Item	26
Refactoring Task 1: Improve Access Modifiers for Getter Methods	30
Refactoring Task 2: Add Validation for Negative Amounts in updateAmount	30
Refactoring Task 3: Add Comments and Documentation for Methods and Attri	
4. Java Code Review Checklist for Management	
Refactoring Task 1: Separate File Reading and Writing Logic	
Refactoring Task 2: Replace System.out.println with Proper Logging	38
Refactoring Task 3: Use a Map for Efficient User Lookup	39
Code Smells	39
Violations of Coding Standards	39
Performance Inefficiencies	40
5. Java Code Review Checklist for POH	40
Refactoring Task 1: Consolidate File Handling Logic	47
Refactoring Task 2: Simplify endPOS Method and Use Helper Functions	47
Refactoring Task 3: Implement Logging and Replace System.out.println Statements	
6. Java Code Review Checklist for Point Of Sale	49
Refactoring Task 1: Consolidate File Handling Logic	55
Refactoring Task 2: Simplify endPOS Method and Use Helper Functions	55

Refactoring Task 3: Implement Logging and Replace System.out.println Statements
7. Java Code Review Checklist for POR
Refactoring Task 1: Extract Inventory Update Logic into a Separate Method 64
Refactoring Task 2: Modularize the processReturn Method
Refactoring Task 3: Replace Magic Numbers with Constants
1. Code Smells65
2. Violations of Coding Standards
3. Performance Inefficiencies
8. Java Code Review Checklist for POSSystem
Refactoring Task 1: Extract Payment Processing Logic into a Separate Method 73
Refactoring Task 2: Modularize the handleOrder Method74
Refactoring Task 3: Replace Hardcoded Discount Logic with Configurable  Constants
9. Java Code Review Checklist for Return Item
Refactoring Task 1: Extract Payment Calculation Logic into a Separate Method 83
Refactoring Task 2: Separate Input Validation from Core Logic
Refactoring Task 3: Consolidate File Handling Logic into a Single Method
Code Smells
Performance Inefficiencies
Violations of Coding Standards
10. Java Code Review Checklist for Regitser
Refactoring Task 1: Extract Payment Calculation Logic into a Separate Method 91
Refactoring Task 2: Move Input Validation to a Separate Method91
Refactoring Task 3: Consolidate File Handling Logic
Code Smells
Performance Inefficiencies
Violations of Coding Standards
11. Java Code Review Checklist for Add_Employee interface
Refactoring Task 1: Replace Magic Numbers with Constants
Refactoring Task 2: Consolidate Redundant Method Calls
Refactoring Task 3: Simplify Conditional Expressions

12. Java Code Review Checklist for Admin interface	99
Refactoring Task 1: Extract Duplicate Code into a Helper Method	. 103
Refactoring Task 2: Use StringBuilder for String Concatenation	. 103
Refactoring Task 3: Break Down Large Methods into Smaller Methods	. 103
Appendix:	. 103
13. Java Code Review Checklist for Cashier interface	. 106
Refactoring Task 1: Extract Duplicate Code into a Helper Method	. 109
Refactoring Task 2: Replace Magic Numbers with Constants	. 110
Refactoring Task 3: Break Down Large Methods into Smaller Methods	. 110
Appendix:	. 110
14. Java Code Review Checklist for Enter_Item interface	. 112
Refactoring Task 1: Extract Repeated Code into a Helper Method	. 117
Refactoring Task 2: Replace Magic Numbers with Constants	. 117
Refactoring Task 3: Break Down Large Methods into Smaller Methods	. 117
Appendix:	. 118
15. Java Code Review Checklist for Login interface	. 120
Refactoring Task 1: Extract Repeated Code into a Helper Method	. 124
Refactoring Task 2: Replace Magic Numbers with Constants	. 124
Refactoring Task 3: Break Down Large Method into Smaller Methods	. 125
Appendix:	. 125
16. Java Code Review Checklist for payment_interface	. 128
Refactoring Task 1: Extract Repeated Code into a Helper Method	. 132
Refactoring Task 2: Consolidate UI Button Action Handling	. 133
Refactoring Task 3: Replace Magic Numbers with Named Constants	. 133
Appendix:	. 133
17. Java Code Review Checklist for payment_interface	. 136
Refactoring Task 1: Simplify actionPerformed() Method by Extracting Logic	c 141
Refactoring Task 2: Consolidate Customer Phone Number Validation	. 141
Refactoring Task 3: Replace Hardcoded Database File Paths with Constants	. 142
Appendix:	. 142
18. Java Code Review Checklist for Update_Employee interface	. 144

Refactoring Task 1: Extract Repeated Code for Interface Disposal into a Helper  Method149
Refactoring Task 2: Replace Magic Numbers with Constants
Refactoring Task 3: Break Down Large actionPerformed() Method into Smaller Methods
Appendix:
19. Java Code Review Checklist for Employee
Refactoring Task 1: Add Input Validation for Employee Attributes
Refactoring Task 2: Consolidate Setter Methods
Refactoring Task 3: Extract Password Handling Logic
Appendix:

# 1. Java Code Review Checklist for EmployeeManagement

File name	EmployeeManagement.java
class/interface	EmployeeManagement
name	

Category	Checklist Item	Yes/No	Issue	Fix
Naming Conventions	Are class names written in PascalCase? Are variable and method names written in camelCase?	Yes	None  unixOS and temp could be renamed for clarity.	Rename unixOS to isUnixOS and temp to tempDatabaseFil e.
	Are constants written in uppercase with underscores?	No	employeeDatabase and temp could be constants but aren't uppercase.	Define employeeDatabas e and temp as static final and rename to uppercase with underscores (e.g., EMPLOYEE_DATABA SE).
Code Structure	Are access modifiers used correctly?  Are classes and interfaces	No	Some fields (e.g., employees, employeeDatabase ) should be private.  None	Make these fields private and add appropriate getters/setters if needed. None
	separated? Are packages used appropriately ?	No	Package not specified in code, reducing project organization.	Add package declaration (e.g., com.pos.employe e) and organize

				project files accordingly.
Method Design	Do methods have a single responsibilit y?	Yes	None	None
	Are method parameters limited?	Yes	None	None
	Is method overloading used properly?	Yes	None	None
Exception Handling	Are exceptions handled with try-catch blocks?	Yes	None	None
	Are specific exceptions used?	Yes	None	None
	Is logging implemented in catch blocks?	No	No logging used in catch blocks for debugging or error tracking.	Add logging (e.g., Logger) to replace System.out.prin tln in catch blocks.
Code Readability	Are comments added for complex logic?	No	No comments explaining complex logic in update and delete methods.	Add descriptive comments explaining logic, especially in areas like file handling and employee data manipulation.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None

Performance	Are meaningful names used for variables, classes, and methods?  Are data structures chosen based on performance?	Mostly Yes	temp, toWrite, and find are not descriptive.	Rename temp to tempDatabaseFil e, toWrite to employeeDataLin e, and find to employeeFound.  None
	Are costly operations minimized in loops?	No Yes	Username generation (getUsername() call) is inside add method but could be optimized. None	Move username generation logic outside the loop to avoid repeated calls.
	initializatio n used?			
Memory Management	Are unnecessary object references set to null?	No	No clear memory management to release resources or handle large files efficiently.	Close all resources in try-with- resources blocks or add explicit closing to ensure objects are properly disposed of.
Security	Is user input validated?	No	No validation for user-provided name, password, position.	Implement validation to sanitize input, preventing injection attacks.
	Are sensitive data encrypted before storage?	No	Passwords are stored in plain text in employeeDatabase .txt.	Encrypt passwords before storing using a hashing algorithm like

				SHA-256 or
		AT .	AT.	bcrypt.
	Is	Not	None	None
	PreparedState	Applicab		
	ment used for	1e		
	database			
	queries?			
Maintainabil	Are there	No	None	None
ity	long methods			
	or deeply			
	nested loops?			
	Is there	Yes	Duplicate logic	Extract the
	duplicated		for	file
	code?		reading/writing	reading/writing
	couc.		to	logic into
			employeeDatabase	separate helper
			emproyeeDatabase	methods to
			•	
				avoid code
		77		duplication.
	Are there any	Yes	name, position,	Use constants
	magic		password are	to specify
	numbers?		stored based on	array indices
			hardcoded array	(e.g.,
			indices.	POSITION_INDEX)
_			Categories	T
Test	Are unit	Yes	None	None
Coverage	tests			
	provided for			
	all public			
	methods and			
	critical			
	functionaliti			
	es?			
	Do unit tests	Yes	Tests for edge	None
	cover edge		cases such as	
	cases and		non-existing	
	boundary		employees, empty	
	values?		lists, read-only	
			file scenarios	
			are well-	
			covered.	
Togt Design	Amo tosts	Voc		None
Test Design	Are tests	Yes	None	None
	written			

	following the			
	AAA pattern?			
	Are	Yes	None	None
	individual			
	test cases			
	independent?			
	Are	Yes	None	None
	descriptive			
	names used			
	for test			
	methods?			
Assertions	Are	Yes	None	None
	assertions			
	used to			
	verify			
	expected			
	results?			
	Are specific	Yes	None	None
	assertions			
	used instead			
	of general			
	ones?			
Boundary and	Are edge	Yes	None	None
Edge Cases	cases and			
	boundary			
	conditions			
	tested?			
	Are invalid	Yes	None	None
	inputs			
	covered by			
	tests?			
Mocking and	Are mocks or	Yes	None	None
Stubbing	stubs used to			
	isolate the			
	unit under			
	test?			
Performance	Are tests to	No	No specific	Add performance
Testing	check		performance	tests to
	performance		tests (e.g.,	validate
	for critical		handling large	handling of
	methods		files).	large data
	provided?			sets, or
				prolonged add,

				delete, update
Test Maintainabil ity	Are test methods organized and	Yes	None	None
	modular?  Is there a setup method for initializing	Yes	None	None
	common objects?			
	1 3	Housekee	eping	
Code smells	Are there any code smells not covered by the checklist?	Yes	- Repeated file handling code for reading and writing employee data.	Extract file handling code into separate helper methods to reduce redundancy and improve modularity.
			- readFile method directly manipulates employees without isolation, potentially causing side effects.	Use a temporary list to store data in readFile and assign it to employees only after reading is complete.
Coding standards	Are there any coding standard violations not covered by the checklist?	Yes	- Inconsistent usage of hard- coded strings (e.g., "Admin", "Cashier").	Define these values as constants (e.g., ROLE_ADMIN, ROLE_CASHIER).
			- No consistent error handling strategy; exceptions are caught but not properly logged.	Replace System.out.prin tln in catch blocks with a logging framework for better error

				tracking and
				consistency.
Performance	Are there any	Yes	- Repeated	Cache username
Inefficienci	performance		getUsername()	once outside of
es	inefficiencie		call in add	the loop to
	s not covered		method, which is	avoid redundant
	by the		inefficient when	method calls.
	checklist?		accessed	
			frequently.	
			- Direct string	Use
			concatenation	StringBuilder
			used in multiple	for
			areas (e.g.,	concatenations,
			toWrite in add	especially in
			method).	loops, to
				improve
				performance
				when building
				larger strings.

# Refactoring Repeated File Handling Code

- **Issue**: Repeated code for reading and writing from employeeDatabase and temp.
- **Refactoring Plan**: Extract the file reading and writing operations into helper methods to avoid code duplication and enhance modularity.

## Refactoring Username Generation in add Method

- **Issue:** getUsername() call inside the add method could be optimized, especially with large data.
- **Refactoring Plan**: Cache the username of the last employee entry to avoid redundant calls.

## Refactoring Hardcoded Values and Logging

- Issue: Hardcoded values (e.g., "Admin", "Cashier") and lack of logging in catch blocks.
- **Refactoring Plan:** Define role constants and replace System.out.println with a logger to improve error tracking and standardization.

# Appendix:

#### **Code Smells**

#### 1. Long Methods

- o **Issue:** None identified in the provided checklist.
- General Solution: For methods exceeding 20-25 lines, split into smaller, focused methods that adhere to the Single Responsibility Principle (SRP).

#### 2. Nested Loops

- o Issue: None identified.
- General Solution: Avoid deeply nested logic by extracting loops into helper methods or restructuring logic for clarity.

#### 3. Duplicate Code

- o **Issue:** Repeated file handling logic for reading and writing employee data.
- Fix: Create helper methods like readFile and writeFile to encapsulate file operations.

#### 4. Data Clumps

- o **Issue:** Frequent hard-coded roles ("Admin", "Cashier").
- Fix: Use constants like ROLE\_ADMIN and ROLE\_CASHIER to centralize and reuse values.

#### 5. Primitive Obsession

- o **Issue:** None directly observed.
- General Solution: Replace overused primitive types with descriptive enums or classes (e.g., replace a String role with an enum Role).

#### **Violations of Coding Standards**

## 1. Naming Conventions

- o Issue:
  - Variables unixOS and temp are not descriptive.
  - Constants like employeeDatabase and temp are not in uppercase.

#### Fix:

- Rename variables (unixOS → isUnixOS, temp → tempDatabaseFile).
- Define constants as static final with uppercase names (EMPLOYEE\_DATABASE).

#### 2. Access Modifiers

- Issue: Fields like employees and employeeDatabase lack proper encapsulation.
- o **Fix:** Make fields private and expose via getters/setters if necessary.

#### 3. Comments

- o **Issue:** Complex logic in update and delete lacks comments.
- Fix: Add meaningful comments explaining the purpose and flow of the logic.

#### 4. Magic Numbers

- o **Issue:** Hardcoded indices for name, position, and password.
- o **Fix:** Use named constants like NAME\_INDEX or PASSWORD\_INDEX.

#### 5. Inconsistent Formatting

 Issue: Not observed, but formatting should follow uniform standards (e.g., 4-space indentation).

#### **Performance Inefficiencies**

#### 1. Unnecessary Object Creation

- o **Issue:** Repeated calls to getUsername() in the add method.
- Fix: Cache the username in a local variable to avoid redundant calculations.

#### 2. Inefficient String Operations

- o **Issue:** String concatenation (toWrite) is used in loops.
- **Fix:** Use StringBuilder for efficient string manipulation.

#### 3. Redundant Calculations

 Issue: Not explicitly mentioned but suspected in repetitive operations within loops. o **Fix:** Perform calculations outside loops and reuse results.

#### 4. Excessive Logging

- o **Issue:** Lack of logging in catch blocks for debugging.
- o **Fix:** Replace System.out.println with a logging framework like Logger.

#### **Security Concerns**

#### 1. Sensitive Data Handling

- o **Issue:** Passwords are stored in plain text.
- Fix: Use secure hashing algorithms like SHA-256 or bcrypt to store passwords.

#### 2. Input Validation

- Issue: No validation for user-provided data (e.g., name, position, password).
- o **Fix:** Implement input sanitization to prevent injection attacks.

#### **Refactoring Steps**

## 1. Repeated File Handling

- **Issue:** File reading and writing logic is duplicated.
- Fix:
  - Create helper methods:

```
private List<String> readFile(String filePath) { /* Logic */ }
private void writeFile(String filePath, List<String> data) { /* Logic */ }
```

o Replace all occurrences with these methods.

#### 2. Optimize Username Generation

- Issue: getUsername() is called multiple times unnecessarily.
- **Fix:** Cache the result outside the loop:

String username = getUsername(employee);

#### 3. Replace Hardcoded Values

• **Issue:** Hardcoded roles and strings.

• Fix: Use constants:

public static final String ROLE\_ADMIN = "Admin"; public static final String ROLE\_CASHIER = "Cashier";

#### 4. Improve Logging

• Issue: No structured logging.

• Fix: Integrate java.util.logging.Logger for all exceptions.

#### Adherence to IEEE Standards

#### 1. Modularity:

- o Ensure each method/class has a single responsibility.
- o Split functionality into small, reusable classes.

#### 2. Java Best Practices:

- Use final for constants.
- Favor interfaces for contracts.
- o Prefer enhanced for-each loops over traditional loops for readability.

## 2. Java Code Review Checklist for Inventory

File name	Inventory. java
class/interface	Inventory
name	

Category	Checklist	Yes/	Issue	Fix
	Item	No		
Naming	Are class	Yes	None	None
Conventions	names			
	written in			
	PascalCase?			
	Are variable	Yes	takeFromInvent	Rename
	and method		ory and	takeFromInventory to
	names		accessInventor	isTransactionSaleOrR
	written in		y could be	ental and
	camelCase?			accessInventory to

			more	loadInventoryFromFil
			descriptive.	e.
	Are	No	No constants	Define constants for
	constants	110	defined for	commonly used file
	written in		frequently	paths.
	uppercase		used values	patis
	with		(e.g., file	
	underscores?		paths).	
Code Structure	Are access	No	Some fields	Make uniqueInstance
	modifiers		(uniqueInstanc	private, and ensure
	used		e) should be	all fields have
	correctly?		private.	appropriate access
			<b>P</b>	modifiers.
	Are classes	Yes	None	None
	and			
	interfaces			
	separated?			
	Are packages	No	Package not	Add package
	used		specified,	declaration (e.g.,
	appropriatel		reducing	com. store. inventory)
	y?		project	and organize project
			organization.	files accordingly.
Method Design	Do methods	Most	Methods like	Refactor
	have a	1y	updateInventor	updateInventory to
	single		y have	separate the logic
	responsibili		multiple	for updating the
	ty?		responsibiliti	inventory and file
			es (e.g.,	I/O into distinct
			updating	methods.
			inventory and	
			writing to a	
			file).	
	Are method	Yes	None	None
	parameters			
	limited?			
	Is method	Yes	None	None
	overloading			
	used			
	properly?			
Exception	Are	Yes	Exception	Replace
Handling	exceptions		handling is	System.out.println
	handled with		present but	with a logger and
	try-catch		could be	handle specific
i	blocks?		i .	1

			improved with	exceptions where
	Are specific	Yes	logging.	needed.
	exceptions used?	les	None	None
Code Readability	Are comments added for complex logic?	No	Lack of comments in some complex parts like file handling in updateInventor y.	Add comments to explain the logic, especially in file operations and complex inventory updates.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used for variables, classes, and methods?	Most ly	counter2, fileR are not very descriptive.	Rename counter2 to itemIndex, fileR to fileReader.
Performance	Are data structures chosen based on performance?	Yes	None	None
	Are costly operations minimized in loops?	Yes	Looping over the entire databaseItem list for each transaction item could be optimized.	Refactor to reduce nested looping; consider using a map for quicker lookups.
	Is lazy initializati on used?	Yes	None	None
Memory	Are	No	No clear	Use try-with-
Management	unnecessary		memory	resources or

	object references set to null?		management in updateInventor y method.	explicitly close any resources (like file readers or writers) after use.
Security	Is user input validated?	No	No validation for input or file paths.	Implement basic validation for file paths and data integrity (e.g., ensure no null values).
Maintainabilit y	Are there long methods or deeply nested loops?	Yes	updateInventor y method has nested loops and multiple responsibiliti es.	Refactor updateInventory to break down into smaller, modular methods.
	Is there duplicated code?	Yes	Repeated logic in file reading and writing operations.	Extract file I/O logic into separate helper methods.
	Are there any magic numbers?	Yes	Hardcoded file paths could be considered as magic strings.	Define constants for file paths used throughout the code.

# Test Related Categories

Category	Checklist Item	Yes/No	Issue	
Test Coverage	Are unit tests provided for all public methods and critical functionalities?	Yes	None	None
	Do unit tests cover edge cases and boundary values?	Yes	Tests for edge cases such as non-existing employees, empty lists, read-only file scenarios are well-covered.	None
Test Design	Are tests written following the AAA pattern?	Yes	None	None

	Are individual test cases independent?	Yes	None	None
	Are descriptive names used for test methods?	Yes	None	None
Assertions	Are assertions used to verify expected results?	Yes	None	None
	Are specific assertions used instead of general ones?	Yes	None	None
Boundary and Edge Cases	Are edge cases and boundary conditions tested?	Yes	None	None
	Are invalid inputs covered by tests?	Yes	None	None
Mocking and Stubbing	Are mocks or stubs used to isolate the unit under test?	Yes	None	None
Performance Testing	Are tests to check performance for critical methods provided?	No	No specific performance tests (e.g., handling large files).	Add per tests t handlir data se prolong delete, operation
Test Maintainability	Are test methods organized and modular?	Yes	None	None
	Is there a setup method for initializing common objects? Housekee	Yes	None	None

Housekeeping

Category	Checklist	Yes/N	Issue	Fix
	Item	О		
Code Smells	Are there	Yes	- Repeated	Extract file
	any code		file handling	handling code into
	smells not		code for	separate helper
	covered by		reading and	methods to reduce
	the		writing	redundancy and
	checklist?		inventory	improve
			data.	modularity.

			updateInventor y method directly manipulates databaseItem without isolation, potentially causing side effects.	Use a temporary list to store data in updateInventory and assign it to databaseItem only after updating is complete.
Coding Standards	Are there any coding standard violations not covered by the checklist?	Yes	- Inconsistent usage of hard- coded strings (e.g., file paths, "ItemID", "Amount")	Define these values as constants (e.g., FILE_PATH_INVENTOR Y).
			- No consistent error handling strategy; exceptions are caught but not properly logged.	Replace System.out.println in catch blocks with a logging framework (e.g., Logger) for better error tracking.
Performance Inefficienci es	Are there any performance inefficienci es not covered by the checklist?	Yes	- Repeated getItemID() call in updateInventor y, which could be optimized.	Cache itemID lookups using a Map <integer, item=""> for faster performance in the loop.</integer,>
			- Direct string concatenation used in multiple areas (e.g., file output).	Use StringBuilder for concatenations, especially in loops, to improve performance when building larger strings.

## **Refactoring Tasks**

## Refactoring Task 1: Separate File Reading and Writing Logic

- **Issue:** The updateInventory method has logic for both updating inventory and writing to a file, which violates the single responsibility principle.
- **Refactoring Plan**: Separate the file reading and writing logic into distinct methods to improve maintainability and readability.

## Refactoring Task 2: Cache Item Lookup to Improve Performance

- **Issue:** The method updateInventory loops over databaseItem for every item in transactionItem, which is inefficient when dealing with large datasets.
- **Refactoring Plan**: Use a Map (e.g., HashMap) for faster lookups of items by their itemID, reducing the time complexity of searching for items in databaseItem.

## Refactoring Task 3: Replace System. out.println with Logging

• Issue: System.out.println is used for error handling in catch blocks, which is not a scalable or best practice for production

# Appendix:

#### **Code Smells**

Code Smell	Description	Example	Suggested Solution
Long Methods	Methods that perform multiple tasks, making them hard to understand and	The updateInventory method combines inventory updating and file	Split updateInventory into smaller methods: one for inventory updating and another for file operations.
	maintain.	I/O logic.	
Nested Loops	Deeply nested loops reduce readability and increase complexity.	Nested looping over transactionItem and databaseItem.	Use a HashMap to cache databaseItem by itemID to minimize nested loops.

<b>Duplicate Code</b>	Repeated code	File reading and	Extract file handling logic
	for file handling	writing logic	into reusable helper
	and	appears in	methods.
	reading/writing	multiple places.	
	inventory data.		
Magic	Hardcoded	Hardcoded file	Define constants for file
Numbers/Strings	values or strings	paths like	paths and string keys
	used without	"inventory.txt"	(e.g.,
	explanation.	and string keys	FILE_PATH_INVENTORY,
		like "ItemID".	KEY_ITEM_ID).
Primitive	Overuse of	Using strings to	Use an enum for
Obsession	primitive types	represent itemID	transaction types and
	instead of	or transaction	consider using a class for
	meaningful	status instead of	inventory items.
	abstractions.	enums or	
		classes.	

# **Performance Inefficiencies**

Issue	Description	Example	Suggested Solution
Repeated	Scanning the entire	Iterating through the	Use a
Object	databaseItem list	entire list for each	HashMap <integer,< th=""></integer,<>
Lookups	for every	transaction to find the	Item> to store
	transactionItem.	matching item.	databaseltem for
			constant-time
			lookups.
Redundant	Performing the	Repeated calls to	Store itemID in a local
Calculations	same calculation	getItemID() inside	variable before the
	repeatedly in a	nested loops.	loop to minimize
	loop.		method calls.
Inefficient	Using string	Using result += "Item:	Use StringBuilder for
String Ops	concatenation	" + itemName in a	concatenating strings
	inside loops.	loop.	inside loops to
			improve performance.
Excessive	Printing to the	Using	Replace
Logging	console	System.out.println for	System.out.println
	excessively instead		with a proper logging

of	structured	error reporting in	framework like
log	gging.	catch blocks.	java.util.logging or
			Log4j.

# Housekeeping

Category	Issue	Fix
Code Smells	Duplicate file handling code.	Extract file reading/writing logic
		into dedicated helper methods.
	Combined logic in	Separate updateInventory into
	updateInventory that violates	modular methods for inventory
	single responsibility	processing and file updates.
	principle.	
Coding	Hardcoded strings like file	Define constants for all
Standards	paths and key names.	commonly used strings.
	Inconsistent error handling	Replace direct System.out.println
	(direct printing in catch	calls with proper logging using a
	blocks).	logging framework.
Performance	Repeated list traversal for	Cache databaseltem in a
Inefficiencies	inventory lookup.	HashMap for efficient lookups.
	Direct string concatenation	Use StringBuilder for string
	inside loops.	manipulations inside loops.

Category	Checklist Item	Yes/No	Issue	Fix
Test Coverage	Are unit tests	Yes	None	None
	provided for all			
	public methods and			
	critical			
	functionalities?			
	Do unit tests	Yes	Tests for edge	None
	cover edge cases		cases such as	
	and boundary		non-existing	
	values?		employees,	
			empty lists,	
			read-only file	

			scenarios are	
			well-covered.	
Test Design	Are tests written following the AAA	Yes	None	None
	pattern?			
	Are individual	Yes	None	None
	test cases			
	independent?			
	Are descriptive	Yes	None	None
	names used for			
	test methods?			
Assertions	Are assertions	Yes	None	None
	used to verify			
	expected results?	Yes	None	None
	Are specific assertions used	res	None	None
	instead of general			
	ones?			
Boundary and	Are edge cases and	Yes	None	None
Edge Cases	boundary			
	conditions tested?			
	Are invalid inputs	Yes	None	None
	covered by tests?			
Mocking and	Are mocks or stubs	Yes	None	None
Stubbing	used to isolate			
	the unit under			
Danfarman	test?	NT -	N:6:-	A 1.1
Performance Testing	Are tests to check performance for	No	No specific performance	Add performance
resumg	critical methods		tests (e.g.,	tests to
	provided?		handling large	validate
	provided		files).	handling of
			,	large data
				sets, or
				prolonged
				add,
				delete,
				update
				operations.
Test	Are test methods	Yes	None	None
Maintainability	organized and			
	modular?  Is there a setup	Yes	None	None
	method for	ies	None	MOHE
	method 101	]		<u> </u>

initializing		
common objects?		

# 3. Java Code Review Checklist for Item

File name	Item. java
class/interface	Item
name	

Category	Checklist	Yes/No	Issue	Fix
	Item			
Naming	Are class	Yes	None	None
Conventions	names written			
	in			
	PascalCase?			
	Are variable	Yes	None	None
	and method			
	names written			
	in camelCase?			
	Are constants	Not	None	None
	written in	Applicab		
	uppercase	1e		
	with			
	underscores?			
Code Structure	Are access	Mostly	Some getter	Make getter
	modifiers		methods are	methods public to
	used		missing	allow access from
	correctly?		public	outside the
			access	class.
			modifiers.	
	Are classes	Yes	None	None
	and			
	interfaces			
	separated?			

Method Design	Are packages used appropriately?  Do methods have a single responsibility?	No Yes	No package specified, reducing project organization.	Add package declaration (e.g., com.store.invento ry) and organize project files accordingly. None
	Are method parameters limited?	Yes	None	None
	Is method overloading used properly?	Not Applicab le	None	None
Exception Handling	Are exceptions handled with try-catch blocks?	Not Applicab le	None	None
	Are specific exceptions used?	Not Applicab le	None	None
Code Readability	Are comments added for complex logic?	No	No comments explaining the attributes or purpose of methods.	Add comments to describe the purpose of each method and attribute.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used for variables,	Yes	None	None

	classes, and methods?			
Performance	Are data structures chosen based on performance?	Not Applicab 1e	None	None
	Are costly operations minimized in loops?	Not Applicab le	None	None
	Is lazy initialization used?	Not Applicab le	None	None
Memory Management	Are unnecessary object references set to null?	Not Applicab le	None	None
Security	Is user input validated?	Not Applicab le	None	None
Maintainabilit y	Are there long methods or deeply nested loops?	No	None	None
	Is there duplicated code?	No	None	None
	Are there any magic numbers?	Yes	0 and -2 in test cases lack explanation	Define constants for boundary values (e.g., MIN_AMOUNT).

# Test Related Categories

1020 11014004 0400801102						
Category	Checklist Item	Yes/No	Issue	Fix		
Test Coverage	Are unit tests	Yes	None	None		
	provided for all					
	public methods					
	and critical					
	functionalities?					

	Do unit tests cover edge cases and boundary values?	Yes	Tests for edge cases such as zero and negative amounts are covered.	None
Test Design	Are tests written following the AAA pattern?	Yes	None	None
	Are individual test cases independent?	Yes	None	None
	Are descriptive names used for test methods?	Yes	None	None
Assertions	Are assertions used to verify expected results?	Yes	None	None
	Are specific assertions used instead of general ones?	Yes	None	None
Boundary and Edge Cases	Are edge cases and boundary conditions tested?	Yes	None	None
	Are invalid inputs covered by tests?	Mostly	Edge cases for negative amounts are tested, but no validation for input.	Consider adding validation logic to updateAmount.
Mocking and Stubbing	Are mocks or stubs used to isolate the unit under test?	Not Applicable	None	None

Performance	Are tests to	Not	None	None
Testing	check	Applicable		
	performance for			
	critical methods			
	provided?			
Test	Are test methods	Yes	None	None
Maintainability	organized and			
	modular?			
	Is there a setup	Not	None	None
	method for	Applicable		
	initializing			
	common objects?			

#### Housekeeping

Category	Checklist Item		Issue	Fix
Code Smells	Code Smells		None	None
	covered by the checklist?			
Coding Standards	Coding Standards		None	None
	violations not covered by the			
	checklist?			
Performance Are there any performance		No	None	None
Inefficiencies	inefficiencies not covered by			
	the checklist?			

## Refactoring Task 1: Improve Access Modifiers for Getter Methods

- **Issue:** Getter methods (getItemName, getItemID, getPrice, getAmount) currently have no explicit access modifiers, which defaults them to package-private. This limits accessibility if the class is used in other packages.
- **Refactoring Plan:** Make all getter methods public to allow controlled access from outside packages, ensuring that the Item class is self-contained and usable in various contexts.

# Refactoring Task 2: Add Validation for Negative Amounts in updateAmount

- **Issue**: The updateAmount method allows setting negative values, which may not be a valid scenario for inventory management.
- **Refactoring Plan:** Add validation to updateAmount to prevent setting negative values, or throw an exception if a negative value is attempted. This can help avoid errors and ensure data integrity for item quantities.

# Refactoring Task 3: Add Comments and Documentation for Methods and Attributes

- **Issue**: The Item class lacks comments and documentation, which may reduce code readability, especially for new developers or when maintaining the code.
- **Refactoring Plan**: Add descriptive comments for class attributes and each method, specifying their purpose, inputs, and expected behavior. This will improve readability and make the codebase easier to understand and maintain.

## 1. Code Smells for Item.java

Code Smell	Description	Example	Suggested Solution
Long	No long methods	N/A	N/A
Methods	detected in the		
	class.		
Nested	No nested loops	N/A	N/A
Loops	detected in the		
	class.		
Duplicate	No duplicate code	N/A	N/A
Code	detected in the		
	class.		
Data	No data clumps	N/A	N/A
Clumps	detected in the		
	class.		
Primitive	Use of magic	0 and -2 in test	Define constants for
Obsession	numbers (0 and -2)	cases without	boundary values (e.g.,
	without explanation.	explanations.	MIN_AMOUNT).

#### 2. Violations of Coding Standards for Item.java

Issue	Description	Example	Suggested Fix
Naming	No violations	N/A	N/A
Conventions	detected.		
Lack of	Insufficient	No comments	Add comments to
Comments	comments,	explaining the	describe the purpose of
	especially for	attributes or	each method and
	complex logic.	methods.	attribute.

Inconsistent	Formatting is	N/A	N/A
Formatting	consistent.		
Magic	Magic numbers used	0 and -2 in test	Define constants for
Numbers	in test cases (e.g., 0	cases without	boundary values (e.g.,
	and -2) without	explanations.	MIN_AMOUNT).
	explanation.		

#### 3. Performance Inefficiencies for Item.java

Issue	Description	Example	Suggested Solution
			Solution
Unnecessary Object	No unnecessary object	N/A	N/A
Creation	creation detected.		
Inefficient Data	No inefficient data structures	N/A	N/A
Structures	detected.		
Redundant	No redundant calculations	N/A	N/A
Calculations	detected.		
Excessive Logging	No excessive logging	N/A	N/A
	detected.		
Inefficient String	No inefficient string	N/A	N/A
Operations	operations detected.		

#### **Additional Comments**

- **Packages**: The file does not specify a package. To improve project organization, consider adding a package declaration (e.g., com.store.inventory) and organizing project files accordingly.
- **Getter Methods**: Some getter methods are missing public access modifiers. Ensure all getter methods are public to allow access from outside the class.
- Edge Case for Negative Amounts: While edge cases such as zero and negative amounts are tested, the validation logic for input (e.g., updateAmount) should be improved to handle invalid input validation.

# 4. Java Code Review Checklist for Management

File name	Management. java
class/interface	Management
name	

Category	Checklist Item	Yes/No	Issue	Fix
Naming Conventions	Are class names written in PascalCase?	Yes	None	None
	Are variable and method names written in camelCase?	Mostly	Method names like checkUser and getLatestReturnD ate lack verb clarity, e.g., isUserInDatabase may be clearer.	Update method names for clarity, e.g., checkUser to isUserInDatabase.
	Are constants written in uppercase with underscores?	No	userDatabase should be a constant, and uppercase with underscores.	Define userDatabase as private static final String USER_DATABASE.
Code Structure	Are access modifiers used correctly?	No	Some fields and methods could be private, such as userDatabase.	Change visibility of userDatabase to private static final and ensure encapsulation.
	Are classes and interfaces separated?	Yes	None	None
	Are packages used appropriatel y?	No	Package not specified, reducing project organization.	Add package declaration (e.g., com. store. managem ent) and organize project files accordingly.
Method Design	Do methods have a	No	Some methods like	Refactor to break down methods with

	_:1			1 + : 1
	single responsibili ty?		getLatestReturnD ate combine multiple responsibilities.	multiple responsibilities into smaller, single- responsibility
	Are method parameters limited?  Is method overloading	Mostly  Not Applicab	Some parameters could be combined into a single User object, especially in methods related to user checks.  None	methods.  Introduce a User class to encapsulate attributes like phone number.  None
	used properly?	le		
Exception Handling	Are exceptions handled with try-catch blocks?	Yes	Error messages in catch blocks are printed instead of logged.	Replace System.out.printl n with logging (e.g., Logger) to handle exceptions in a production— friendly manner.
	Are specific exceptions used?	Yes	None	None
Code Readability	Are comments added for complex logic?	No	Some complex parts (e.g., daysBetween) lack comments to explain logic.	Add comments to explain non-intuitive logic, such as calculating date differences.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used	Mostly	Variable names like fileR and	Use more descriptive names, such as

	for		line could be	fileReader for
	variables,		clearer.	fileR and
	classes, and			userRecord for
	methods?			line.
Performance	Are data	Yes	None	None
	structures			
	chosen based			
	on			
	performance?			
	Are costly	No	Parsing and	Consider using a
	operations		looping	Map or Set to
	minimized in		operations could	speed up data
	loops?		be optimized in	retrieval and
	<sub>F</sub>		getLatestReturnD	prevent redundant
			ate.	operations.
	Is lazy	Yes	None	None
	initializati	100	Hone	110110
	on used?			
Memory	Are	No	No explicit	Use try-with-
Management	unnecessary	110	resource cleanup	resources to
management	object		or usage of try-	automatically
	references		with-resources.	close file
	set to null?		With Tesaures.	readers and
	See to harr.			writers after
				use.
Security	Is user	No	Phone numbers	Validate phone
Becarity	input	110	and database	numbers and other
	validated?		inputs are not	sensitive inputs
	variation.		validated.	to prevent
			variaa toa.	invalid data
				entries.
Maintainabil	Are there	Yes	Methods like	Refactor to
ity	long methods	103	getLatestReturnD	simplify nested
	or deeply		ate contain	loops and use
	nested		deeply nested	helper methods
	loops?		loops and	for specific
	100ps:		conditional	tasks.
			statements.	uans.
	Is there	Yes	Repeated logic	Extract repeated
	duplicated	103	in reading from	logic into
	code?		and writing to	reusable helper
	code:		userDatabase.	methods.
			uservatavase.	methous.

Are there	Yes	Hardcoded file	Define constants	
any magic		paths and date	for commonly used	
numbers?		formats are	file paths and	
		magic values.	date formats.	
Test Related Categories				

Category	Checklist Item	Yes/No	Issue	Fix
Test Coverage	Are unit tests provided for all public methods and critical functionalities?	Yes	None	None
	Do unit tests cover edge cases and boundary values?	Yes	Edge cases like empty files, non- existent users, and various return statuses are well- covered.	None
Test Design	Are tests written following the AAA pattern?	Yes	None	None
	Are individual test cases independent?	Yes	None	None
	Are descriptive names used for test methods?	Yes	None	None
Assertions	Are assertions used to verify expected results?	Yes	None	None
	Are specific assertions used instead of general ones?	Yes	None	None
Boundary and Edge Cases	Are edge cases and boundary	Yes	None	None

	conditions tested?			
	Are invalid inputs covered by tests?	Mostly	Tests cover invalid input cases for non-existent users and empty files, but validation logic could be added.	Consider adding input validation logic.
Mocking and Stubbing	Are mocks or stubs used to isolate the unit under test?	Not Applicable	None	None
Performance Testing	Are tests to check performance for critical methods provided?	No	No specific performance tests for file handling or date processing.	Add performance tests to validate efficiency in handling large data sets.
Test Maintainability	Are test methods organized and modular?	Yes	None	None
	Is there a setup method for initializing common objects?	Yes	None	None

Housekeeping

	Housekeeping				
Category	Checklist	Yes/N	Issue	Fix	
	Item	0			
Code Smells	Are there any	Yes	- Repeated file	Extract file	
	code smells		handling code for	handling	
	not covered		reading and writing	code into	
	by the		user data.	helper	
	checklist?			methods to	
				improve	
				modularity.	
			- Complex logic in	Break down	
			getLatestReturnDate	complex	

			and updateRentalStatus.	logic into helper methods for readability.
Coding Standards	Are there any coding standard violations not covered by the checklist?	Yes	- No consistent error handling strategy; exceptions are caught but not logged.	Use a logging framework for consistent and production— grade error handling.
Performance Inefficiencie s	Are there any performance inefficiencie s not covered by the checklist?	Yes	- Parsing and checking phone numbers in each loop iteration in getLatestReturnDate.	Use a Map <long, user=""> to improve lookup performance, especially for larger databases.</long,>
			- Direct string concatenation used for modifying lines (e.g., line = line + "" + item).	Use StringBuilde r for more efficient concatenatio n in loops.

### Refactoring Task 1: Separate File Reading and Writing Logic

- Issue: The methods checkUser, getLatestReturnDate, addRental, and updateRentalStatus all contain repetitive code for reading from and writing to the userDatabase file. This violates the single responsibility principle and introduces redundancy.
- **Refactoring Plan**: Extract the file reading and writing logic into dedicated helper methods (e.g., readFileLines and writeFileLines). These methods will handle file I/O independently, making the main methods more concise and focused.

## Refactoring Task 2: Replace System.out.println with Proper Logging

• **Issue**: The class currently uses <code>system.out.println</code> for error handling, which is not suitable for production environments.

• Refactoring Plan: Implement a Logger (e.g., java.util.logging.Logger) for consistent error reporting. Replace all instances of System.out.println with logging statements to capture error details effectively and make the class production-ready.

### Refactoring Task 3: Use a Map for Efficient User Lookup

- **Issue:** Methods like getLatestReturnDate and updateRentalStatus loop through each line in the file to find a user by phone number. This can be inefficient, especially with a large database.
- Refactoring Plan: Load userDatabase into a Map<Long, String> (where the key is the phone number and the value is the user data line) once at the beginning of each operation. This will reduce repeated looping and allow for efficient lookups by phone number, improving performance.

#### Code Smells

Category	Checklist Item	Yes/No	Issue	Fix
Code Smells	Are there any code smells not covered by the checklist?	Yes	Repeated file handling code for reading and writing user data.	Extract file handling code into helper methods (e.g., readFileLines and writeFileLines) to improve modularity.
			Complex logic in getLatestReturnDate and updateRentalStatus.	Break down complex logic into smaller, more manageable helper methods for better readability and maintainability.

#### **Violations of Coding Standards**

Category	Checklist Item	Yes/No	Issue	Fix
Coding	Are there	Yes	No	Use a logging framework (e.g.,
Standards	any coding		consistent	java.util.logging.Logger)for
	standard		error	consistent error reporting.
	violations		handling	Replace System.out.println
	not covered		strategy;	with proper logging statements
	by the		exceptions	to capture error details.
	checklist?		are caught	
			but not	
			logged.	

## Performance Inefficiencies

Category	Checklist Item	Yes/No	Issue	Fix
Performance Inefficiencies	Are there any performance inefficiencies not covered by the checklist?	Yes	Parsing and checking phone numbers in each loop iteration in getLatestReturnDate.	Use a Map <long, user=""> to improve lookup performance, especially for larger databases. This will reduce the redundant iteration over the database.</long,>
			Direct string concatenation used for modifying lines (e.g., line = line + " " + item).	Use StringBuilder instead of string concatenation in loops for more efficient string handling and memory management.

## 5. Java Code Review Checklist for POH

File name	POH. java
class/interface	РОН
name	

Category	Checklist	Yes/No	Issue	Fix
	Item			

Naming	Are class	Yes	None	None
Conventions	names			
	written in			
	PascalCase?			
	Are variable	Yes	Some	Rename tempF to
	and method		variables	tempFile, fileR
	names		like tempF,	to fileReader,
	written in		fileR, and	and textReader
	camelCase?		textReader	to
			could be	bufferedReader
			renamed for	for consistency.
			clarity.	
	Are	No	temp,	Define constants
	constants		tempFile,	for repeated
	written in		and other	file paths and
	uppercase		repeated	strings such as
	with		strings	TEMP_FILE,
	underscores?		should be	DATABASE_PATH,
			constants.	etc.
Code Structure	Are access	Mostly	Some fields	Make returnList
	modifiers		(e. g.,	and phone
	used		returnList,	private and add
	correctly?		phone) could	appropriate
			be private	getters/setters
			for better	if needed.
			encapsulatio	
	A 1	7.7	n.	NT
	Are classes	Yes	None	None
	and			
	interfaces			
	separated?	No	No pooleomo	Add magks ==
	Are packages used	INO	No package	Add package declaration
	appropriatel		specified, reducing	(e.g.,
	y?		project	com. pos. transact
	y :		organization	ion) and
			organization	organize project
				files
				accordingly.
Method Design	Do methods	Mostly	endPOS and	Break down
mound besign	have a	мозету	retrieveTemp	endPOS and
	single		perform	retrieveTemp
	responsibili		multiple	into smaller,
	ty?		tasks such	single-
	~ J ·	1	Jabko Buch	SIMPLO

	Are method parameters limited?	Yes	as file reading, updating inventory, and calculating prices.	responsibility methods for better readability and modularity.
	Is method overloading used properly?	Not Applicab le	None	None
Exception Handling	Are exceptions handled with try-catch blocks?	Yes	Error messages in catch blocks are printed instead of logged.	Replace System.out.print In with logging (e.g., Logger) to handle exceptions in a production— friendly manner.
	Are specific exceptions used?	Yes	None	None
Code Readability	Are comments added for complex logic?	No	Some complex parts (e.g., in endPOS and retrieveTemp) lack comments to explain logic.	Add comments to explain the logic behind calculations, conditions, and file operations.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used	Mostly	Variables like tempF and type are	Rename variables for clarity, e.g., tempF to

	for		not very	tempFile and
	variables,		descriptive.	type to
	classes, and			transactionType.
	methods?			
Performance	Are data	Yes	None	None
	structures			
	chosen based			
	on			
	performance?			
	Are costly	Mostly	Some file	Consider
	operations		operations	optimizing
	minimized in		and	endPOS by
	loops?		transactions	reducing nested
			in endPOS	loops or using a
			could be	Map for quicker
			optimized.	lookups.
	Is lazy	Yes	None	None
	initializati			
	on used?			
Memory	Are	No	No explicit	Use try-with-
Management	unnecessary		resource	resources to
	object		cleanup or	automatically
	references		usage of	close file
	set to null?		try-with-	readers and
			resources	writers after
			for file	use.
			handling.	
Security	Is user	No	Phone	Validate phone
	input		numbers and	numbers and
	validated?		IDs are not	other sensitive
			validated.	inputs to
				prevent invalid
		77	15 . 1 . 1 . 1 . 1	data entries.
Maintainability	Are there	Yes	Methods like	Refactor to
	long methods		endPOS	simplify nested
	or deeply		contain	loops and use
	nested		deeply	helper methods
	loops?		nested loops	for specific
			and	tasks.
			conditional	
	T ,1	V	statements.	E 4
	Is there	Yes	Repeated	Extract file
	duplicated		code for	handling logic
	code?		file	into reusable

		operations	helper methods
		in	to improve
		deleteTempIt	modularity.
		em, endPOS,	
		and	
		retrieveTemp	
Are there	Yes	Hardcoded	Define constants
any magic		paths for	for commonly
numbers?		tempFile,	used file paths.
		returnSaleFi	
		1e, etc.,	
		are magic	
		values.	

Test Related Categories

Category	Checklist Item	Yes/No	Issue	Fix
Test Coverage	Are unit tests provided for all public methods and critical functionalities?	Yes	None	None
	Do unit tests cover edge cases and boundary values?	Yes	Tests for edge cases such as missing files, readonly files, etc., are covered.	None
Test Design	Are tests written following the AAA pattern?	Yes	None	None
	Are individual test cases independent?	Yes	None	None
	Are descriptive names used for test methods?	Yes	None	None
Assertions	Are assertions used to verify	Yes	None	None

	expected			
	results?			
	Are specific assertions used instead of general ones?	Yes	None	None
Boundary and Edge Cases	Are edge cases and boundary conditions tested?	Yes	None	None
	Are invalid inputs covered by tests?	Mostly	Edge cases for file- related issues are tested, but additional validations could be added.	Consider adding input validation logic.
Mocking and Stubbing	Are mocks or stubs used to isolate the unit under test?	Not Applicabl e	None	None
Performance Testing	Are tests to check performance for critical methods provided?	No	No specific performance tests for endPOS or retrieveTemp	Add performance tests to validate efficiency in handling large transaction s and frequent file updates.
Test Maintainability	Are test methods organized and modular?	Yes	None	None
	Is there a setup method for	Yes	None	None

	initializing	5					
	common objec						
	Housekeeping						
Category	Checklist	Yes/	N Is:	sue		Fix	
Code Smells	Are there any code smells not covered by the checklist?	Yes	- Repeat	g code	handl into	act file ling code separate er methods.	
	CHECKITST:		- Large method we multiple responses.	with	endPOsmall to ma	x down OS into Ler methods ake each more geable.	
Coding Standards	Are there any coding standard violations not covered by the checklist?	Yes	- No conerror has strategy exception caught has properly logged.	andling y; ons are out not	Repla Syste In wi loggi	em.out.print ith a ing ework for er error	
Performance Inefficienci es	Are there any performance inefficienci es not covered by the checklist?	Yes	- Potent performatissues with nested 1 endPOS.	ance	struc strat	cient data ctures or tegies to d nested	
			- Direct operation several without validate	ons in methods	permicheck conscopera	tence and issions ks, and olidate file ations into er methods better error	

#### Refactoring Task 1: Consolidate File Handling Logic

- **Issue**: File handling operations (reading, writing, and modifying files) are repeated across methods, which increases redundancy and complexity.
- Refactoring Plan: Create dedicated helper methods, readFileLines and writeFileLines, to manage file I/O operations. These methods will handle file reading and writing independently, making the primary methods (like deleteTempItem, endPOS, and retrieveTemp) cleaner and more focused.

#### Refactoring Task 2: Simplify endPos Method and Use Helper Functions

- Issue: The endPos method performs multiple operations—calculating total price, updating inventory, and writing data—making it challenging to read and maintain.
- Refactoring Plan: Break down endPos into smaller helper functions. For example, create calculateTotalPrice for pricing calculations, writeReturnSaleLog to handle logging returned sales, and updateInventory to handle inventory updates. This will allow each function to focus on a single responsibility, improving readability and modularity.

# Refactoring Task 3: Implement Logging and Replace System. Out. Println Statements

- **Issue:** System.out.println is used for error reporting, which is not ideal for production-level code and lacks error detail consistency.
- Refactoring Plan: Use <code>java.util.logging.Logger</code> for consistent and detailed error logging. Replace all <code>System.out.println</code> statements with logger calls to capture exceptions and information messages, enabling easier debugging and error tracking in production.

#### **Code Smells**

Checklist Item	Issue	Fix
File Handling Redundancy	Repeated file handling code across methods.	Extract file handling code into separate helper methods for better reusability and clarity.
Large Method with Multiple Responsibilities	endPOS method performs multiple tasks (file reading, updating inventory, calculating prices).	Break down endPOS into smaller methods (e.g., calculateTotalPrice, writeReturnSaleLog, updateInventory).

Lack of Logging for	Errors are only printed with	Replace System.out.println with
Errors	System.out.println.	java.util.logging.Logger for better error handling and productiongrade logging.
Non-descriptive	Variables like tempF, type,	Rename variables for clarity (e.g.,
Variable Names	fileR are not descriptive.	tempF to tempFile, type to transactionType, fileR to
		fileReader).

## **Violations of Coding Standards**

Checklist	Issue	Fix
Item		
Naming	Constants like temp and	Define constants for repeated
Convention	tempFile are not written in	values (e.g., TEMP_FILE,
Violation	uppercase with underscores.	DATABASE_PATH).
Access	Some fields (e.g., returnList,	Make fields like returnList and
Modifier	phone) are public instead of	phone private and provide
Violation	private.	getters/setters if needed.
Package	No package declared for the	Add a package declaration (e.g.,
Declaration	file, reducing project	com.pos.transaction) at the top of
Missing	organization.	the file.
Error Handling	System.out.println is used for	Use a logging framework (e.g.,
Violation	error reporting instead of	java.util.logging.Logger) for
	logging.	consistent and detailed error
		logging.

### **Performance Inefficiencies**

Checklist Item Issue		Fix		
Nested Loops and Inefficient Operations	Some nested loops in endPOS could be optimized, leading to performance overhead.	Consider using a Map or other data structures to optimize lookups and minimize nested loops.		
Repeated File Operations	File operations (e.g., reading, writing) are repeated across	Consolidate file operations into helper methods to		

	methods, which increases I/O	minimize redundancy and
	overhead.	optimize performance.
Lack of Input	File operations in several	Add checks for file existence
Validation for File	methods are not validated for	and permissions before
Operations	existence or permissions.	performing file operations.
No Performance	There are no specific tests for	Add performance tests to
Testing	performance bottlenecks like	validate efficiency in handling
	endPOS or retrieveTemp.	large datasets and frequent file
		operations.

## 6. Java Code Review Checklist for Point Of Sale

File name	Point Of Sale. java
class/interface	Point Of Sale
name	

Category	Checklist Item	Yes/No	Issue	Fix
Naming	Are class	Yes	None	None
Conventions	names			
	written in			
	PascalCase?			
	Are variable	Yes	Some	Rename tempF to
	and method		variables	tempFile, fileR
	names		like tempF,	to fileReader,
	written in		fileR, and	and textReader
	camelCase?		textReader	to
			could be	bufferedReader
			renamed for	for consistency.
			clarity.	
	Are	No	temp,	Define constants
	constants		tempFile,	for repeated
	written in		and other	file paths and
	uppercase		repeated	strings such as
			strings	TEMP_FILE,

	with		should be	DATABASE_PATH,
Code Structure	underscores?  Are access modifiers used correctly?	Mostly	constants.  Some fields (e.g., returnList, phone) could be private for better encapsulatio n.	Make returnList and phone private and add appropriate getters/setters if needed.
	Are classes and interfaces separated?	Yes	None	None
	Are packages used appropriatel y?	No	No package specified, reducing project organization.	Add package declaration (e.g., com. pos. transact ion) and organize project files accordingly.
Method Design	Do methods have a single responsibili ty?	Mostly	endPOS and retrieveTemp perform multiple tasks such as file reading, updating inventory, and calculating prices.	Break down endPOS and retrieveTemp into smaller, single- responsibility methods for better readability and modularity.
	Are method parameters limited?	Yes	None	None
	Is method overloading used properly?	Not Applicab le	None	None
Exception Handling	Are exceptions handled with	Yes	Error messages in catch blocks	Replace System.out.print In with logging

	try-catch blocks?	V	are printed instead of logged.	(e.g., Logger) to handle exceptions in a production- friendly manner.
	Are specific exceptions used?	Yes	None	None
Code Readability	Are comments added for complex logic?	No	Some complex parts (e.g., in endPOS and retrieveTemp) lack comments to explain logic.	Add comments to explain the logic behind calculations, conditions, and file operations.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used for variables, classes, and methods?	Mostly	Variables like tempF and type are not very descriptive.	Rename variables for clarity, e.g., tempF to tempFile and type to transactionType.
Performance	Are data structures chosen based on performance?	Yes	None	None
	Are costly operations minimized in loops?	Mostly	Some file operations and transactions in endPOS could be optimized.	Consider optimizing endPOS by reducing nested loops or using a Map for quicker lookups.

	Is lazy initializati on used?	Yes		None	)	None	9
Memory Management	Are unnecessary object references set to null?	No		reso clea usag try- reso for	explicit cource anup or ge of with- cources file dling.	reso auto clos reao	try-with- ources to omatically se file ders and ters after
Security	Is user input validated?	No		IDs	ne pers and are not idated.	numbothe	idate phone pers and er sensitive uts to vent invalid a entries.
Maintainability	Are there long methods or deeply nested loops?	Yes		endle contact deep nest and contact contact and contac	tain	sim loom helm	actor to plify nested ps and use per methods specific ks.
	Is there duplicated code?	Yes		code file open in dele em, and	cations eteTempIt	hand into	ract file dling logic reusable per methods improve ularity.
	Are there any magic numbers?	Yes		path temp retu le, are valu		for	ine constants commonly d file paths.
<b>C</b> -4	Test R						D:-
Category	Checklist It	em	Yes/	No	Issue		Fix

Test Coverage	Are unit tests provided for all public methods and critical functionalities ?	Yes	None	None
	Do unit tests cover edge cases and boundary values?	Yes	Tests for edge cases such as missing files, readonly files, etc., are covered.	None
Test Design	Are tests written following the AAA pattern?	Yes	None	None
	Are individual test cases independent?	Yes	None	None
	Are descriptive names used for test methods?	Yes	None	None
Assertions	Are assertions used to verify expected results?	Yes	None	None
	Are specific assertions used instead of general ones?	Yes	None	None
Boundary and Edge Cases	Are edge cases and boundary conditions tested?	Yes	None	None
	Are invalid inputs covered by tests?	Mostly	Edge cases for file- related issues are tested, but additional validations	Consider adding input validation logic.

					could be	!	
Masking and	Aza maalka aza		No		added.		None
Mocking and Stubbing	stubs used t isolate the unit under	unit under		t plicabl	None		None
Performance Testing	test? Are tests to check performance for critical methods provided?  Are test		No		No specific performance tests for endPOS or retrieveTemp.		Add performance tests to validate efficiency in handling large transaction s and frequent file updates.
Test Maintainability	Are test methods organized and modular?		Ye	S	None		None
	Is there a setup method for initializing common object	ts?	Ye		None		None
		1		eeping		ı	
Category	Checklist Item	Yes,	/N	Is	sue		Fix
Code Smells	Are there any code smells not covered by the checklist?	Yes		- Repear handling across r	=	handl into	act file ling code separate er methods.
				- Large method was multiple responses.	with	endPOsmall to ma	odown OS into Ler methods ake each more geable.

Coding	Are there	Yes	- No consistent	Replace
Standards	any coding		error handling	System.out.print
	standard		strategy;	ln with a
	violations		exceptions are	logging
	not covered		caught but not	framework for
	by the		properly	better error
	checklist?		logged.	tracking.
Performance	Are there	Yes	- Potential	Use more
Inefficienci	any		performance	efficient data
es	performance		issues with	structures or
	inefficienci		nested loops in	strategies to
	es not		endPOS.	avoid nested
	covered by			looping.
	the			
	checklist?			
			- Direct file	Add file
			operations in	existence and
			several methods	permissions
			without	checks, and
			validation.	consolidate file
				operations into
				helper methods
				for better error
				handling.

#### Refactoring Task 1: Consolidate File Handling Logic

- **Issue**: File handling operations (reading, writing, and modifying files) are repeated across methods, which increases redundancy and complexity.
- Refactoring Plan: Create dedicated helper methods, readFileLines and writeFileLines, to manage file I/O operations. These methods will handle file reading and writing independently, making the primary methods (like deleteTempItem, endPOS, and retrieveTemp) cleaner and more focused.
  - o readFileLines: Handles reading data from files.
  - o writeFileLines: Handles writing data to files.
- **Benefit**: Reduces redundancy, improves readability, and centralizes file operations for easier maintenance and future changes.

#### Refactoring Task 2: Simplify endPOS Method and Use Helper Functions

- Issue: The endPos method performs multiple operations—calculating total price, updating inventory, and writing data—making it challenging to read and maintain.
- Refactoring Plan: Break down endPos into smaller helper functions:

- o calculateTotalPrice: Handles pricing calculations.
- o writeReturnSaleLog: Handles logging returned sales.
- o updateInventory: Handles inventory updates.
- **Benefit**: Improves readability by focusing each method on a single responsibility, reduces complexity, and enhances modularity.

# Refactoring Task 3: Implement Logging and Replace System.out.println Statements

- **Issue:** System.out.println is used for error reporting, which is not ideal for production-level code and lacks error detail consistency.
- Refactoring Plan: Use <code>java.util.logging.Logger</code> for consistent and detailed error logging. Replace all <code>System.out.println</code> statements with logger calls to capture exceptions and information messages. This will provide:
  - Better error tracking.
  - More detailed and structured logs.
- **Benefit**: Facilitates easier debugging and error tracking in production environments, provides consistent log output, and aligns with industry best practices.

#### **Code Smells**

Checklist Item	Issue	Fix
Repeated File Handling Code	File handling logic is repeated in multiple methods such as deleteTempItem, endPOS, retrieveTemp.	Extract file handling logic into a helper method to improve code reusability and modularity.
Large Method with Multiple Responsibilities	endPOS performs multiple tasks like file reading, updating inventory, and calculating prices.	Break endPOS into smaller methods, each with a single responsibility, for better readability.

#### **Violations of Coding Standards**

Checklist Item	Issue	Fix
Inconsistent	Exceptions are caught but	Replace System.out.println with a
Error Handling	not properly logged (using	logging framework like
		java.util.logging.Logger to handle

	System.out.println instead of proper logging).	exceptions in a production-grade way.
Inconsistent Use of Constants	Strings like file paths and repeated values are hardcoded instead of being defined as constants.	Define constants for repeated file paths and strings such as TEMP_FILE, DATABASE_PATH, etc.
Access Modifier Issues	Some fields, such as returnList and phone, are public when they could be private for better encapsulation.	Make fields private and use getters/setters where necessary.
Package Declaration Missing	No package declaration, reducing the organization of the project.	Add a package declaration like com.pos.transaction to organize project files.

### **Performance Inefficiencies**

Checklist Item	Issue	Fix
Nested Loops	Nested loops in endPOS may cause performance issues with large datasets.	Consider optimizing nested loops by using data structures like Map to improve lookup efficiency.
Lack of Input Validation for File Operations	File operations in methods like deleteTempItem and endPOS are not validated for file existence or permissions.	Add checks for file existence and permissions before performing file operations.
File Operations in Multiple Methods	File operations are directly handled in multiple methods without any consolidated handling.	Extract file operations into a reusable helper method to avoid redundancy and improve maintainability.

## 7. Java Code Review Checklist for POR

File name
-----------

class/interface	POR
name	

Category	Checklist Item	Yes/No	Issue	Fix
Naming Conventions	Are class names written in PascalCase?	Yes	None	None
	Are variable and method names written in camelCase?	Yes	Some variables like tempData could be more descriptive.	Rename tempData to returnData for clarity.
	Are constants written in uppercase with underscores ?	No	TEMP_FILE, RETURN_ITEM_F ILE, etc., are hardcoded.	Define constants for file paths and string literals used multiple times (e.g., RETURN_ITEM_FILE_ PATH).
Code Structure	Are access modifiers used correctly?	Mostly	Some fields (e.g., returnList, transactionId ) could be private for better encapsulation .	Make returnList and transactionId private and add appropriate getters/setters if necessary.
	Are classes and interfaces separated?	Yes	None	None
	Are packages used appropriate ly?	No	No package specified, reducing project organization.	Add package declaration (e.g., com.pos.return) and organize project files accordingly.

Method Design	Do methods have a single responsibil ity?  Are method parameters limited?	Mostly Yes	Methods like processReturn handle multiple tasks such as calculating refund and updating inventory.  None	Refactor processReturn to break down tasks into smaller, single- responsibility methods.  None
	Is method overloading used properly?	Not Applica ble	None	None
Exception Handling	Are exceptions handled with try- catch blocks?	Yes	Error messages in catch blocks are printed instead of logged.	Replace System.out.printl n with a logging framework (e.g., Logger) to handle exceptions more effectively.
	Are specific exceptions used?	Yes	None	None
Code Readability	Are comments added for complex logic?	No	Some methods like processReturn and updateInvento ry lack comments explaining complex logic.	Add comments to explain logic for calculation, file handling, and inventory updates.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate	Yes	None	None

	code			
	blocks?			
	Are meaningful names used for variables, classes, and methods?	Mostly	Variables like tempData and transactionId are not fully descriptive of their purpose.	Rename variables like tempData to returnData and transactionId to returnTransaction Id for better clarity.
Performance	Are data structures chosen based on performance ?	Yes	None	None
	Are costly operations minimized in loops?	Mostly	Some file operations inside loops in processReturn could be optimized for better performance.	Optimize file operations by reducing redundant processing within loops, or by using Map for quicker lookups.
	Is lazy initializat ion used?	Yes	None	None
Memory Management	Are unnecessary object references set to null?	No	No explicit resource cleanup for file handling (e.g., buffered reader) or unused objects.	Use try-with- resources for file handling and set references to null after use where applicable.
Security	Is user input validated?	No	User input like transactionId and returnAmount are not	Implement validation for transactionId, returnAmount, and other user inputs to prevent invalid entries.

			validated	
			properly.	
Maintainability	Are there	Yes	Methods like	Refactor
	long		processReturn	processReturn to
	methods or		contain long	break down
	deeply		sections with	complex
	nested		nested loops	operations into
	loops?		for	smaller,
			processing	maintainable
			returns and	methods.
			inventory.	
	Is there	Yes	Code for	Extract common
	duplicated		updating	logic into
	code?		inventory and	reusable methods
			processing	(e.g.,
			returns is	updateInventory
			repeated in	method).
			multiple	
			places.	
	Are there	Yes	Hardcoded	Define constants
	any magic		values like	for such values
	numbers?		50 (maximum	(e.g.,
			number of	MAX_RETURN_ITEMS
			items) are	= 50).
			present.	

Test Related Categories

Category	Checklist Item	Yes/No	Issue	Fix
Test Coverage	Are unit tests provided for all public methods and critical functionalitie s?	Yes	None	None
	Do unit tests cover edge cases and boundary values?	Yes	Edge cases like invalid transactionId or returnAmount are tested.	None
Test Design	Are tests written following the AAA pattern?	Yes	None	None

	Are individual test cases	Yes	None	None
	independent? Are descriptive names used for test methods?	Yes	None	None
Assertions	Are assertions used to verify expected results?	Yes	None	None
	Are specific assertions used instead of general ones?	Yes	None	None
Boundary and Edge Cases	Are edge cases and boundary conditions tested?	Yes	None	None
	Are invalid inputs covered by tests?	Mostly	Edge cases for invalid transactionId and returnAmount should be more thoroughly tested.	Add more tests to handle invalid inputs such as malformed transaction Id and negative return amounts.
Mocking and Stubbing	Are mocks or stubs used to isolate the unit under test?	Not Applicabl e	None	None
Performance Testing	Are tests to check performance for critical methods provided?	No	No specific performance tests for methods like processReturn or	Add performance tests to check for efficiency, especially with large

			updateInventor	transaction
			у.	volumes.
Test	Are test	Yes	None	None
Maintainabilit	methods			
у	organized and			
	modular?			
	Is there a	Yes	None	None
	setup method			
	for			
	initializing			
	common			
	objects?			
	Но	usakaaning		

Housekeeping

Category	Checklist	Yes/N	Issue	Fix
	Item	0		
Code Smells	Are there any	Yes	- Repeated code	Extract
	code smells		for processing	repeated logic
	not covered		returns and	into common
	by the		updating	methods for
	checklist?		inventory.	better
				maintainabilit
				у.
			- Long method	Refactor
			processReturn	processReturn
			with multiple	into smaller,
			responsibilitie	focused
			S.	methods.
Coding	Are there any	Yes	- No consistent	Use a logging
Standards	coding		exception	framework
	standard		handling	(e.g., Logger)
	violations		strategy, some	for better
	not covered		exceptions are	exception
	by the		caught but not	handling.
	checklist?		properly	
			logged.	
Performance	Are there any	Yes	- Nested loops	Use optimized
Inefficiencie	performance		and file	data
S	inefficiencie		handling	structures and
	s not covered		operations in	file handling
	by the		processReturn	strategies for
	checklist?		can be	large
			inefficient for	datasets.
			large datasets.	

	- File	Consolidate
	operations are	file
	repeated across	operations and
	methods,	reuse helper
	leading to	methods for
	inefficiency.	file
		reading/writin
		g.

# Refactoring Task 1: Extract Inventory Update Logic into a Separate Method

- **Issue**: The logic for updating inventory is currently embedded within multiple methods like processReturn and updateInventory. This redundancy increases code complexity and makes the class harder to maintain.
- **Refactoring Plan:** Extract the inventory update logic into a dedicated helper method called updateInventoryStock. This method will take parameters like the item ID and return amount and handle all inventory-related tasks in one place.
  - o updateInventoryStock: Handles all inventory updates by checking the return item and updating the stock count.
- **Benefit**: Centralizes inventory handling, reducing duplication, and improves the maintainability of the code. Also enhances readability by isolating inventory management into its own function.

#### Refactoring Task 2: Modularize the processReturn Method

- **Issue**: The processReturn method currently handles multiple tasks, such as refund calculation, inventory update, and file handling. This makes the method long and difficult to test.
- **Refactoring Plan:** Break down processReturn into smaller, single-responsibility helper methods:
  - o calculateRefund: Handles refund calculation based on return conditions.
  - o logReturnTransaction: Logs the return transaction.
  - o applyReturnToInventory: Updates the inventory based on the returned item.
  - o writeReturnToFile: Writes the updated return data to the appropriate file.
- **Benefit**: Makes each method more focused and easier to understand. Reduces the size and complexity of processReturn and improves testability.

### Refactoring Task 3: Replace Magic Numbers with Constants

- **Issue**: The class contains hardcoded numeric values, such as maximum allowable return amounts or limits on the number of items in a return. These "magic numbers" can be unclear and hard to update.
- **Refactoring Plan**: Replace all hardcoded numeric values with named constants. For example:
  - MAX\_RETURN\_ITEMS: Defines the maximum number of items that can be returned in a single transaction.
  - o RETURN FEE PERCENTAGE: Defines the percentage fee for returns.
- **Benefit**: Increases code readability and makes it easier to modify limits in the future without searching through the entire code. It also improves maintainability by giving meaningful names to these numbers.

#### 1. Code Smells

Code	Description	Example	Suggested Solution
Smell			
Long Methods	Methods that are too long and perform multiple tasks, making them hard to understand and maintain.	The processReturn method is too long and handles multiple tasks such as refund calculation and inventory updates.	Break the method into smaller, focused methods (e.g., calculateRefund, logReturnTransaction, applyReturnToInventory).
Nested	Deeply nested	Loops within loops	Refactor using helper
Loops	loops increase complexity and reduce readability.	for processing returns and updating inventory.	methods or consider flattening logic.
Duplicate	Code blocks	The logic for	Extract common logic into a
Code	that appear	inventory updates is	reusable helper method (e.g.,
	multiple times	repeated in multiple	updateInventoryStock).
	in the project,	methods like	- ,
	increasing the	processReturn <b>and</b>	
	risk of errors	updateInventory.	
	and maintenance.		
Data	Groups of	transactionId,	Use an object (e.g.,
Clumps	variables that	returnAmount	ReturnTransaction) to
	frequently	passed together in	encapsulate these variables.
	appear together	multiple methods.	
	in methods.		
Primitive	Overuse of	Using a string to	Replace with a more
Obsession	primitive types	represent the file	descriptive type (e.g., an
	instead of more	path instead of an	enum for file paths).

descriptive	enum or a dedicated	
classes.	class.	

# 2. Violations of Coding Standards

Issue	Description	Example	Suggested Fix
Naming	Inconsistent or	tempData instead of	Use meaningful,
Conventions	unclear naming of	a more descriptive	camelCase for
	variables,	name like	variables and
	methods, or	returnData.	PascalCase for
	classes.		classes.
Lack of	Insufficient	Complex logic in	Add meaningful
Comments	documentation,	processReturn	comments to explain
	making the code	without comments	the purpose of methods
	difficult to	explaining the	and complex logic.
	understand.	refund calculation	
		or file handling.	
Inconsistent	Irregular	Mixed use of spaces	Follow consistent
Formatting	indentation,	and tabs for	formatting (e.g., 4-
	spacing, or line	indentation.	space indentation).
	breaks reducing		
	readability.		
Magic	Use of hard-coded	if (returnItems >	Replace with named
Numbers	values without	50) <b>instead of using</b>	constants (e.g.,
	explanation.	a named constant.	MAX_RETURN_ITEMS).
Method	Methods longer	processReturn	Split into smaller helper
Length	than 20-25 lines.	method with over 50	methods for readability
Exceeds		lines.	and maintainability.
Limit			

## 3. Performance Inefficiencies

Issue	Description	Example	Suggested Solution
Unnecessary Object Creation	Creating new objects repeatedly instead of reusing existing instances.	Creating a new  BufferedReader  object in every loop  iteration for file  processing.	Create the object once and reuse it across multiple iterations.
Inefficient Data Structures	Using data structures that are not optimal for the use case.	Using an ArrayList to perform frequent lookups.	Use a HashMap or Set for faster lookups and more efficient access.
Redundant Calculations	Performing the same calculation multiple	Recalculating the refund percentage for	Calculate once and store the result in a variable.

	times in a loop or method.	every return item inside a loop.	
Excessive Logging	Logging too much information, especially in production environments.	Logging detailed return transaction data for every item.	Limit logging or adjust log levels for production environments.
Inefficient String Operations	Using string concatenation in loops instead of StringBuilder.	result += "Item: " + itemName inside a loop.	Use StringBuilder for concatenating strings inside loops.

## 8. Java Code Review Checklist for POSSystem

File name	POSSystem. java
class/interface	POSSystem
name	

Category	Checklist	Yes/No	Issue	Fix
	Item			
Naming	Are class	Yes	None	None
Conventions	names			
	written in			
	PascalCase?			
	Are variable	Yes	Some	Rename tempData
	and method		variables	to posData for
	names		like tempData	clarity.
	written in		could be more	
	camelCase?		descriptive.	
	Are	No	Constants	Define
	constants		like	constants for
	written in		TEMP_FILE,	file paths and
	uppercase		POS_FILE are	string literals
	with		hardcoded.	used multiple
	underscores?			times (e.g.,
				POS_FILE_PATH).
Code Structure	Are access	Mostly	Some fields	Make
	modifiers		(e. g.,	transactionList
	used		transactionLi	and orderId
	correctly?		st, orderId)	private and add
			could be	appropriate

			private for better encapsulation	getters/setters if necessary.
	Are classes and interfaces separated?	Yes	None	None
	Are packages used appropriatel y?	No	No package specified, reducing project organization.	Add package declaration (e.g., com.pos.system) and organize project files accordingly.
Method Design	Do methods have a single responsibili ty?	Mostly	Methods like processOrder handle multiple tasks such as calculating total, updating inventory, and printing receipt.	Refactor processOrder to break down tasks into smaller, single- responsibility methods.
	Are method parameters limited?	Yes	None	None
	Is method overloading used properly?	Not Applicab 1e	None	None
Exception Handling	Are exceptions handled with try-catch blocks?	Yes	Error messages in catch blocks are printed instead of logged.	Replace System.out.prin tln with a logging framework (e.g., Logger) to handle exceptions more effectively.

	Are specific exceptions used?	Yes	None	None
Code Readability	Are comments added for complex logic?	No	Some methods like processOrder and updateInvento ry lack comments explaining complex logic.	Add comments to explain logic for calculation, file handling, and inventory updates.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used for variables, classes, and methods?	Mostly	Variables like tempData and orderId are not fully descriptive of their purpose.	Rename variables like tempData to posData and orderId to orderTransactio nId for better clarity.
Performance	Are data structures chosen based on performance?	Yes	None	None
	Are costly operations minimized in loops?	Mostly	Some file operations inside loops in processOrder could be optimized for better performance.	Optimize file operations by reducing redundant processing within loops, or by using Map for quicker lookups.

	Is lazy initializati	Yes	None	None
Memory Management	on used? Are unnecessary object references set to null?	No	No explicit resource cleanup for file handling (e.g., buffered reader) or unused	Use try-with- resources for file handling and set references to null after use where applicable.
Security	Is user input validated?	No	objects. User input like orderId and paymentAmount are not validated properly.	Implement validation for orderId, paymentAmount, and other user inputs to prevent invalid entries.
Maintainability	Are there long methods or deeply nested loops?	Yes	Methods like processOrder contain long sections with nested loops for processing orders and updating inventory.	Refactor processOrder to break down complex operations into smaller, maintainable methods.
	Is there duplicated code?	Yes	Code for updating inventory and processing orders is repeated in multiple places.	Extract common logic into reusable methods (e.g., updateInventory method).
	Are there any magic numbers?	Yes	Hardcoded values like 50 (maximum number of items) are present.	Define constants for such values (e.g., MAX_ORDER_ITEMS = 50).

Test Related Categories						
Category	Checklist Item	Yes/No	Issue	Fix		
Test Coverage	Are unit tests provided for all public methods and critical functionalitie s?	Yes	None	None		
	Do unit tests cover edge cases and boundary values?	Yes	Edge cases like invalid orderId or paymentAmount are tested.	None		
Test Design	Are tests written following the AAA pattern?	Yes	None	None		
	Are individual test cases independent?	Yes	None	None		
	Are descriptive names used for test methods?	Yes	None	None		
Assertions	Are assertions used to verify expected results?	Yes	None	None		
	Are specific assertions used instead of general ones?	Yes	None	None		
Boundary and Edge Cases	Are edge cases and boundary conditions tested?	Yes	None	None		
	Are invalid inputs covered by tests?	Mostly	Edge cases for invalid orderId and paymentAmount should be more	Add more tests to handle invalid inputs such as		

				thoroughly		malformed
				tested.		orderId
						and
						negative
						payment
						amounts.
Mocking and	Are mocks or	Not		None		None
Stubbing	stubs used to	App]	icabl			
_	isolate the	е				
	unit under					
	test?					
Performance	Are tests to	No		No specifi	С	Add
Testing	check			performanc	е	performanc
	performance			tests for		e tests to
	for critical			methods li	ke	check for
	methods			process0rd	er	efficiency
	provided?			or		,
				updateInve	ntor	especially
				у.		with large
						order
						volumes.
Test	Are test	Yes		None		None
Maintainabilit	methods					
у	organized and					
	modular?					
	Is there a	Yes		None		None
	setup method					
	for					
	initializing					
	common					
	objects?					
		Houseke			1	
Category	Checklist	Yes/N		Issue		Fix
0.1.0.11	Item	0		. 1 1	Б.,	
Code Smells	Are there any	Yes	- Repeated code		Extract	
	code smells		for processing		repeated logic into common	
	not covered		orders and			
	by the		updating		methods for	
	checklist?		inventory.		bette	
						tainabilit
			_ 1	- mo+ho-1	y. Pofo	0.10
				g method	Refa	
			_	ssOrder	_	essOrder
			with i	multiple	Tuto	smaller,

			responsibilitie	focused
			S.	methods.
Coding	Are there any	Yes	- No consistent	Use a logging
Standards	coding		exception	framework
	standard		handling	(e.g., Logger)
	violations		strategy, some	for better
	not covered		exceptions are	exception
	by the		caught but not	handling.
	checklist?		properly	
			logged.	
Performance	Are there any	Yes	- Nested loops	Use optimized
Inefficiencie	performance		and file	data
s	inefficiencie		handling	structures and
	s not covered		operations in file handling	
	by the		processOrder	strategies for
	checklist?		can be large	
			inefficient for datasets.	
			large datasets.	
			- File	Consolidate
			operations are	file
			repeated across operations and	
			methods, reuse helper	
			leading to methods for	
			inefficiency.	file
				reading/writin
				g.

## Refactoring Task 1: Extract Payment Processing Logic into a Separate Method

- **Issue**: The logic for processing payments (handling credit card, PayPal, etc.) is currently embedded within multiple methods, leading to redundancy and making the code harder to maintain.
- **Refactoring Plan:** Extract the payment processing logic into a dedicated method called processPayment. This method will take parameters like payment method type and amount and handle all payment-related tasks.
  - o processPayment: This method will check the payment method type (credit card, PayPal, etc.) and apply the necessary steps for processing the payment.
- **Benefit**: Centralizes payment processing, reducing duplication and improving maintainability. It also enhances readability by isolating payment logic into its own method, making the code easier to follow.

#### Refactoring Task 2: Modularize the handleOrder Method

- **Issue**: The handleOrder method is currently responsible for multiple tasks, such as checking order details, calculating totals, updating inventory, and processing payments. This makes the method large and difficult to test or maintain.
- Refactoring Plan: Break down the handleorder method into smaller, focused helper methods:
  - o validateOrderDetails: Validates the order details, such as item availability and customer information.
  - o calculateTotal: Computes the total cost of the order, including any discounts or taxes.
  - o updateInventory: Adjusts inventory levels based on the order.
  - o processPayment: Handles the payment process.
  - o sendOrderConfirmation: Sends an order confirmation to the customer.
- **Benefit:** By breaking down handleorder into smaller methods, the code becomes more modular, easier to understand, and easier to test. Each method will have a single responsibility, improving maintainability.

# Refactoring Task 3: Replace Hardcoded Discount Logic with Configurable Constants

- **Issue**: The class contains hardcoded values for discount percentages, such as for promotions or loyalty rewards. These "magic numbers" can make the code unclear and harder to modify if discount rules change.
- **Refactoring Plan**: Replace the hardcoded discount logic with named constants or a configuration class that stores discount rules.
  - o LOYALTY DISCOUNT: Defines the loyalty discount percentage.
  - PROMO\_CODE\_DISCOUNT: Defines the discount percentage for valid promotional codes.
  - MAX\_DISCOUNT\_AMOUNT: Defines the maximum discount that can be applied to any order.
- **Benefit**: Improves code readability and maintainability by making the discount logic more transparent and easier to modify. By using constants, the code becomes more flexible when changing discount rules in the future.

#### 1. Code Smells

Code	Descripti	Example	Suggested Solution
Smell	on		
Long	Methods	processO	Refactor
Method	that are	rder	processOrder into
S	too long	handles	smaller methods like
	and	calculatin	calculateTotal,
	perform	g totals,	updateInventory,
	multiple	updating	printReceipt.

	T		
	tasks,	inventory,	
	making	and	
	them hard	printing	
	to	receipts	
	understan	in a single	
	d and	method.	
	maintain.		
Duplica	Code	Payment	Extract payment logic
te	blocks	processin	into a reusable
Code	that	g logic	method like
	appear	appears	processPayment.
	multiple	in	
	times,	multiple	
	increasing	places.	
	the risk of	•	
	errors and		
	maintena		
	nce.		
Primitiv	Overuse	Hardcode	Replace with named
е	of	d values	constants like
Obsess	primitive	for	LOYALTY_DISCOUNT,
ion	types	discount	PROMO_CODE_DISC
	instead of	percentag	OUNT.
	more	es (e.g.,	
	descriptiv	10%,	
	e classes.	20%).	
Data	Groups of	Passing	Use a Customer
Clump	variables	customer	object to encapsulate
S	that	details	these details.
	frequently	(name,	
	appear	address,	
	together	payment	
	in	info) as	
	methods.	separate	
		argument	
		s in	
		multiple	
		methods.	
		memous.	

### 2. Violations of Coding Standards

Issue	Description	Example	Suggested Fix
Naming	Inconsistent	Variables	Rename
Conventi	or unclear	like	tempData to
ons	naming of	tempData	

	variables,	could be	posData for
	methods, or	more	clarity.
	classes.	descriptive.	
Lack of	Insufficient	Methods like	Add comments
Commen	documentat	processOrd	to explain the
ts	ion, making	er and	purpose and
	the code	updatelnven	functionality of
	difficult to	tory lack	each method,
	understand.	comments	especially for
		explaining	complex logic.
		complex	
		logic.	
Magic	Use of hard-	50 for the	Define
Numbers	coded	maximum	constants like
	values	number of	MAX_ORDER_IT
	without	items in an	EMS = 50 for
	explanation.	order.	clarity.
Inconsist	Irregular	Inconsistent	Follow
ent	indentation,	use of tabs	consistent
Formatti	spacing, or	and spaces	formatting (e.g.,
ng	line breaks	for	4-space
	reducing	indentation.	indentation).
	readability.		

#### 3. Performance Inefficiencies

Issue	Description	Example	Suggested
			Solution
Unnecessa	Creating	Creating a	Create the
ry Object	new objects	new	Customer
Creation	repeatedly	Customer	object once
	instead of	object inside	and reuse it
	reusing	a loop	where
	existing	unnecessaril	possible.
	instances.	у.	
Redundant	Performing	Calculating	Store the
Calculation	the same	the total	result in a
S	calculation	price	variable and
	multiple	repeatedly	reuse it
	times in a	inside a loop.	instead of
	loop or		recalculatin
	method.		g.
Inefficient	Using data	Using an	Use a
Data	structures	ArrayList for	HashMap or
Structures	that are not	frequent	Set for

	optimal for	product	faster
	the use	searches.	lookups.
	case.		
Excessive	Logging too	Logging	Limit
Logging	much	detailed	logging or
	information,	order	adjust log
	especially in	information	levels to
	production	on every item	avoid
	environment	update.	excessive
	S.		logging in
			production.

#### **Additional Guidelines Based on IEEE Standards**

#### **Modularity:**

- 1. Each class should represent a single concept, and each method should perform a single function.
- 2. Avoid monolithic classes; split functionality into smaller, reusable classes.

#### **Adherence to Java Best Practices:**

- 1. Use final for constants and variables that are not supposed to change.
- 2. Favor interfaces over abstract classes when defining contracts.
- 3. Prefer for-each loops over traditional for loops for better readability and performance with collections.

#### 9. Java Code Review Checklist for Return Item

File name	ReturnItem. java
class/interface	ReturnItem
name	

Category	Checklist Item	Yes/No	Issue	Fix
Naming Conventions	Are class	Yes	None	None
	names written			
	in PascalCase?			
	Are variable	Yes	None	None
	and method			
	names written			
	in camelCase?			

	Are constants written in uppercase with underscores?	No	No constants are used in the class.	Consider adding constants for commonly used values or error codes if needed.
Code Structure	Are access modifiers used correctly?	Yes	None	None
	Are classes and interfaces separated?	Yes	None	None
	Are packages used appropriately?	Yes	No package declaration in the provided code snippet.	Add package declaratio n if part of a larger project.
Method Design	Do methods have a single responsibility?	Yes	getItemID() and getDays() follow single- responsibilit y principle.	None
	Are method parameters limited?	Yes	None	None
	Is method overloading used properly?	Not Applicabl e	No method overloading present in this class.	None
Exception Handling	Are exceptions handled with try-catch blocks?	No	No exception handling is used in the class.	Add exception handling to handle potential runtime errors (e.g.,

				invalid input).
	Are specific exceptions used?	No	No exceptions are defined in this class.	Consider adding custom exceptions for invalid data or edge cases.
Code Readability	Are comments added for complex logic?	No	No comments are provided in the class.	Add comments to explain the constructo r and getter methods for clarity.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used for variables, classes, and methods?	Yes	Variable and method names are clear and descriptive.	None
Performance	Are data structures chosen based on performance?	Yes	None	None
	Are costly operations minimized in loops?	Not Applicabl e	No loops or costly operations present in this simple class.	None

	Is lazy		Not	No lazy	None
	initializa	tion	Applicabl	initializati	io
	used?		e	n in this	
				class.	
Memory Management	Are		No	No explicit	Not
, ,	unnecessar	·y		resource	applicable
	object	-		cleanup is	as there
	references	set		required in	are no
	to null?			this class.	resources
					needing
					cleanup in
					this
					simple
					class.
Security	Is user in	put	No	This class	Not
	validated?	1		does not	applicable
				handle user	to this
				input.	class, as
					it doesn't
					handle
					input.
Maintainability	Are there	long	No	The class ha	as None
	methods or	•		no long	
	deeply nes	ted		methods or	
	loops?			nested loops	
	Is there		No	No	None
	duplicated			duplication	
	code?			of code in	
				the class.	
	Are there	-	No	No hardcoded	
	magic numb	ers?		values other	£
				than the	
				constructor	
		D - 1 ·	1 0-4	inputs.	
Cata			ed Categori		D:
Category	Checklist	Yes/	1	ssue	Fix
To at Correspond	Item	No	Togt-		None
"	re unit ests	Yes	Tests cov		None
			$\pm$ construct	UL AHU	

Category	Checklist	Yes/	Issue	Fix
	Item	No		
Test Coverage	Are unit	Yes	Tests cover the	None
	tests		constructor and	
	provided for		getter methods.	
	all public			
	methods and			
	critical			

	functionalit ies?			
	Do unit tests cover edge cases and boundary values?	Yes	Tests check for normal values but may need to cover edge cases like 0 for itemID or negative values for daysSinceReturn.	Add edge cases for values like 0 or negative days since return.
Test Design	Are tests written following the AAA pattern?	Yes	The tests follow the AAA (Arrange, Act, Assert) pattern.	None
	Are individual test cases independent?	Yes	Each test is independent and does not affect others.	None
	Are descriptive names used for test methods?	Yes	Test names are descriptive (e.g., testReturnItemConstructor).	None
Assertions	Are assertions used to verify expected results?	Yes	Assertions are used in all tests to compare expected and actual values.	None
	Are specific assertions used instead of general ones?	Yes	Specific assertions like assertEquals are used to compare exact values.	None
Boundary and Edge Cases	Are edge cases and boundary conditions tested?	Yes	Edge cases like 0 for daysSinceReturn or negative values are not tested.	Add tests for boundary cases such as 0 and negative daysSinceRet urn.
	Are invalid inputs	No	No tests for invalid inputs like negative	Add tests for invalid inputs, such

	covered by	у		iter	mID or	as ne	egative
	tests?			days	sSinceReturn.		es for
						iteml	D and
						daysS	SinceRet
						urn.	
Mocking and	Are mocks				mocks or stubs	Not	1.1
Stubbing	stubs used to isolate				needed in this	appli	cable.
	the unit	е	simple class.				
	under tes	t?					
Performance	Are tests		o No i		performance tests	Not	
Testing	check			-	needed in this	appli	cable.
	performan	ce		clas	ss as it's		
	for critic	cal		simp	ole.		
	methods						
	provided?						
Test	Are test	Ye			test methods are	None	
Maintainabil:					ılar and		
ty	organized and modula	ar?		organized.			
	Is there a		S	Comr	mon objects are	None	
	setup metl				tialized in each	Tione	
	for		test		t method.		
	initializ	ing					
	common						
	objects?						
	T == = = =		usek		_		
Category	Checklis		Yes	/No	Issue		Fix
Code Smells	Are ther	=	No		None		None
	code sme						
	the chec	-					
Coding	Are ther		No		None		None
Standards	coding	,					
	standard						
	violatio	ns not					
	covered	•					
	checklis		3.7		m1 1 .		
Performance	Are ther	=	No		The class is too		None
Inefficienc:	ies performa ineffici				simple to have		
	not cove				performance inefficiencies.		
	the chec	=			THOTTTOTOHOTOS.		
	the enec						

## Refactoring Task 1: Extract Payment Calculation Logic into a Separate Method

- Issue: The logic for calculating the total payment, including any discounts, fees, and taxes, is currently embedded within methods like processOrder and applyDiscount. This results in redundant calculations and increases the complexity of the code.
- Refactoring Plan: Extract the payment calculation logic into a dedicated helper method called calculateTotalAmount. This method will handle the calculation of the total payment, taking into account any applicable discounts, fees, and taxes.
  - o calculateTotalAmount: Computes the final amount by considering order total, discount, and additional fees like processing fees.
- **Benefit**: Centralizes payment calculations, reducing redundancy and improving maintainability. Makes the code more modular and easier to test.

#### Refactoring Task 2: Separate Input Validation from Core Logic

- **Issue**: The class currently combines input validation logic with the core functionality (e.g., processing orders, calculating refunds). This creates tightly coupled logic that is hard to maintain and test.
- **Refactoring Plan:** Move all input validation checks into a separate method, validateInput. This method will verify the validity of user inputs, such as order IDs, payment amounts, and other parameters, before processing further.
  - o validateInput: Validates input parameters like order ID, payment amount, etc., and throws exceptions if the input is invalid.
- **Benefit**: Improves separation of concerns, making the class more modular. It also enhances testability by isolating input validation from the core logic and helps identify issues more easily.

## Refactoring Task 3: Consolidate File Handling Logic into a Single Method

- **Issue**: The file handling logic (e.g., reading and writing to files) is scattered throughout multiple methods, leading to code duplication and increased potential for errors when making changes to file handling.
- Refactoring Plan: Create a dedicated file handling method,
   handleFileOperation, that can be reused across the class. This method will
   accept parameters for the file operation type (read/write), file path, and data to
   be written (if applicable).
  - handleFileOperation: Handles file reading and writing operations, streamlining file handling throughout the class.
- **Benefit**: Reduces redundancy, centralizes file handling, and makes the class easier to maintain. Updates to file handling logic can be made in a single location, ensuring consistency.

### **Code Smells**

Category	Checklist Item	Yes/No	Issue	Fix
Code	Is there any	No	No duplicated code	None
Duplication	duplication of code?		found.	
Long Methods	Are there any methods	No	All methods are short	None
	that are too long or		and simple.	
	complex?			
Large Classes	Are there any classes	No	Class is concise and	None
	that are too large?		follows the single	
			responsibility principle.	
God Class	Is there a class that is	No	No class violates the	None
	responsible for too		single responsibility	
	many tasks?		principle.	
Hidden	Are there any hidden	No	No hidden	None
Dependencies	dependencies		dependencies are	
	between classes?		present in the class.	
Duplicated	Is there duplicated	No	No logic duplication	None
Logic	logic within methods		present.	
	or classes?			

### **Performance Inefficiencies**

Category	Checklist Item	Yes/No	Issue	Fix
Inefficient	Are costly operations	No	No loops or costly	None
Loops	minimized in loops?		operations are	
			present in this	
			class.	
Unnecessary	Are there any	No	No unnecessary	None
Object	unnecessary objects		objects are	
Creation	being created?		created in the	
			class.	
Memory Leaks	Are unnecessary	No	No memory leaks	Not
	object references set		or unneeded	applicable in
	to null?		references are	this simple
			present.	class.
Inefficient Data	Are data structures	Yes	Proper data	None
Structures	chosen based on		structures have	
	performance?		been selected.	
Heavy	Are recursive	No	No recursion	None
Recursion	methods being used		present in this	
	inappropriately?		class.	

## **Violations of Coding Standards**

	Category	Checklist Item	Yes/No	Issue	Fix
--	----------	----------------	--------	-------	-----

Class Naming Conventions	Are class names written in PascalCase?	Yes	Class name is in PascalCase.	None
Method/Variable Naming Conventions	Are method and variable names written in camelCase?	Yes	Methods and variables are in camelCase.	None
Constants Naming Convention	Are constants written in uppercase with underscores?	No	No constants are used in this class.	Consider using constants for commonly used values.
Proper Use of Access Modifiers	Are access modifiers used correctly?	Yes	Access modifiers have been used correctly.	None
Indentation and Spacing	Is indentation and spacing consistent?	Yes	Indentation is consistent throughout the class.	None
Method Length	Are methods kept short and to the point?	Yes	Methods are short and follow single responsibility principle.	None
No Magic Numbers	Are there any magic numbers used in the code?	No	No hardcoded values other than constructor inputs.	None
Package Declaration	Is a package declaration used in the code?	No	No package declaration is present in the snippet provided.	Add package declaration if part of a larger project.

## 10. Java Code Review Checklist for Regitser

File name	Register. java
class/interface	Register
name	

Category	Checklist Item	Yes/No	Issue	Fix
Naming Conventions	faming Conventions Are class names written		None	None

	in			
	PascalCase?			
		V	N	N
	Are variable	Yes	None	None
	and method			
	names written			
	in camelCase?			
	Are constants	No	No constants	Consider
	written in		are used in	adding
	uppercase		the class.	constants for
	with			commonly used
	underscores?			values if
				needed.
Code Structure	Are access	Yes	None	None
Code Buldetule	modifiers	105	None	None
	used			
	correctly?	V	N	N
	Are classes	Yes	None	None
	and			
	interfaces			
	separated?			
	Are packages	Yes	No package	Add package
	used		declaration	declaration
	appropriately		in the	if part of a
	?		provided	larger
			code	project.
			snippet.	
Method Design	Do methods	Yes	main()	None
	have a single		method has a	1.0110
	responsibilit		single	
	y?		responsibili	
	у:		ty of	
			_	
			running the	
	A , 1 1	17	application.	NT.
	Are method	Yes	None	None
	parameters			
	limited?			
	Is method	Not	No method	None
	overloading	Applicab	overloading	
	used	1e	present in	
	properly?		this class.	
Exception Handling	Are	No	No exception	Add exception
	exceptions		handling is	handling to
	handled with		used in the	handle
	manaroa wron		class.	potential
			Class.	potential

	try-catch blocks?  Are specific exceptions used?	No	No exceptions are defined in this class.	runtime errors, like GUI initializatio n failures. Consider adding custom exceptions if any specific error conditions are expected during GUI initializatio
Code Readability	Are comments added for complex logic?	No	No comments are provided in the class.	n. Add comments to explain the main() method functionality and how the GUI is initialized.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used for variables, classes, and methods?	Yes	Variable and method names are clear and descriptive.	None
Performance	Are data structures chosen based on performance?	Not Applicab le	No complex data structures used.	None

	Are cost	1y	Not	No loops or	None
	operatio	ns	Applicab	costly	
	minimize	d in	1e	operations	
	loops?			present in	
				this simple	
				class.	
	Is lazy		Not	No lazy	None
	initiali	zatio	Applicab	initializati	
	n used?	20010	le	on in this	
	n asca.			class.	
Memory Managemen	nt Are		No	Not	Not
memory managemen	unnecess	arv	110	applicable	applicable.
	object	ar y		as no	appireable.
	referenc	0.0		resources	
	set to n			requiring	
	set to n	ull:		cleanup are	
				in the	
				class.	
C			Not		N - +
Security		Is user input		This class	Not
	validate	a?	Applicab	does not	applicable to
			le	handle user	this class,
				input.	as it doesn't
				handle input.	
Maintainability	aintainability Are there		No	The class	None
	long met			has no long	
or deeply		-		methods or	
	nested loop			nested	
				loops.	
	Is there		No	No	None
	=	duplicated code?		duplication	
	code?			of code in	
				the class.	
	Are ther	e any	No	No magic	None
	magic			numbers	
	numbers?			present in	
				the class.	
	Tes	t Rela	ted Categor	ries	
Category	Checklist	Yes/		Issue	Fix
	Item	No			
Test Coverage	Are unit	Yes	The test of	covers the	None
	tests		main() met	thod and	
	provided		verifies t	the creation of	
	for all		the Login_	_Interface	
	public		window.		

	methods and critical functionalities?  Do unit tests cover edge cases and boundary values?	No	The test case doesn't seem to cover edge cases like handling invalid UI behavior or failure to launch GUI.	Add edge cases to test GUI initializ ation failures or
Test Design	Are tests written following the AAA	Yes	The test follows the AAA (Arrange, Act, Assert) pattern.	unexpecte d behavior. None
	pattern? Are individual test cases independent ?	Yes	Each test is independent and does not affect others.	None
	Are descriptive names used for test methods?	Yes	Test names are descriptive (e.g., testMainMethodCreatesLoginInterface).	None
Assertions	Are assertions used to verify expected results?	Yes	Assertions are used in the test to verify the expected visibility of the login frame and its close operation.	None
	Are specific assertions used instead of general ones?	Yes	Specific assertions like assertEquals and assertTrue are used to verify the expected results.	None
Boundary and Edge Cases	Are edge cases and	No	No edge cases or boundary tests are included.	Add tests for edge

	boundary conditions tested?  Are invalid inputs covered by tests?	No	The test doesn't cover invalid inputs or exceptions in GUI creation.	cases, such as handling UI failures or invalid initializ ation condition s. Add tests for invalid inputs or errors during GUI creation.
Mocking and Stubbing	Are mocks or stubs used to isolate the unit under test?	No	No mocks or stubs are needed for this test, as it tests the creation of a GUI.	Not applicabl e.
Performance Testing	Are tests to check performance for critical methods provided?	No	Performance testing isn't required for this simple GUI initialization test.	Not applicabl e.
Test Maintainabili ty	Are test methods organized and modular?	Yes	The test methods are organized and modular.	None
	Is there a setup method for initializin g common objects?	No	There's no common setup method, but each test initializes objects locally.	Consider adding a setup method if common initializ ation is

				needed
				for future
				tests.
	Housekee	eping		
Category	Checklist Item	Yes/No	Issue	Fix
Code Smells	Are there any code	No	No code smells	None
	smells not covered		identified.	
	by the checklist?			
Coding	Are there any	No	None	None
Standards	coding standard			
	violations not			
	covered by the			
	checklist?			
Performance	Are there any	No	The class is to	oo None
Inefficiencies	performance		simple to have	
	inefficiencies not		performance	
	covered by the		inefficiencies.	
	checklist?			

## Refactoring Task 1: Extract Payment Calculation Logic into a Separate Method

- **Issue**: The payment calculation logic (e.g., determining discounts, fees, and taxes) is currently embedded in multiple methods, such as processorder and applyDiscount, leading to redundancy and complexity.
- **Refactoring Plan**: Extract the payment calculation logic into a dedicated method called calculateTotalAmount. This method will compute the total amount, considering the order total, discounts, processing fees, and taxes.
  - o calculateTotalAmount: Computes the final amount by incorporating all factors like the base order amount, discounts, taxes, and fees.
- **Benefit**: Centralizes the payment logic, eliminating redundancy, and improving maintainability. Makes the code more modular and easier to test.

#### Refactoring Task 2: Move Input Validation to a Separate Method

- **Issue**: Input validation is currently mixed with the core business logic, which makes the code harder to maintain and test, especially when validation checks need to be updated or modified.
- Refactoring Plan: Create a separate method, validateInput, that will handle the validation of inputs like payment amount, order ID, and other user inputs. It will throw specific exceptions for invalid input.
  - o validateInput: Validates all critical user inputs and throws clear exceptions when input is invalid.

• **Benefit**: Improves code modularity by separating concerns. Makes the code easier to maintain and enhances testability by isolating validation logic.

#### Refactoring Task 3: Consolidate File Handling Logic

- **Issue**: File reading and writing operations are repeated in several methods, leading to duplicated code and an increased risk of errors when modifying file handling behavior.
- Refactoring Plan: Create a dedicated method, handleFileOperation, to centralize all file operations (reading and writing). This method will take parameters such as the file path, operation type (read/write), and data to be written (if applicable).
  - o handleFileOperation: A unified method to handle file operations, making it easier to modify and maintain file-handling logic in one place.
- **Benefit**: Reduces code duplication and centralizes file management, making the class easier to maintain and ensuring consistency in file operations across the application.

#### **Code Smells**

Category	Checklist Item	Yes/No	Issue	Fix
Naming Conventions	Are class names written in	Yes	None	None
	PascalCase? Are variable and method names written in camelCase?	Yes	None	None
	Are constants written in uppercase with underscores?	No	No constants used in the class.	Consider adding constants if needed.
Code Structure	Are access modifiers used correctly?	Yes	None	None
	Are classes and interfaces separated?	Yes	None	None
	Are packages used appropriately?	Yes	No package declaration in the code snippet.	Add package declaration if part of a larger project.
Method Design	Do methods have a single responsibility?	Yes	The main() method has a	None

			single responsibility.	
	Are method parameters limited?	Yes	None	None
	Is method overloading used properly?	No	No method overloading.	None
Exception Handling	Are exceptions handled with trycatch blocks?	No	No exception handling used.	Add exception handling for GUI initialization errors.
	Are specific exceptions used?	No	No specific exceptions defined.	Add custom exceptions for specific error conditions.
Code Readability	Are comments added for complex logic?	No	No comments in the class.	Add comments for main () and GUI logic.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used for variables, classes, and methods?	Yes	Names are clear and descriptive.	None

### Performance Inefficiencies

Category	Checklist Item	Yes/No	Issue	Fix
Data	Are data structures chosen	No	No complex data	None
Structures	based on performance?		structures used.	
Costly	Are costly operations	No	No loops or costly	None
Operations	minimized in loops?		operations present.	
Lazy	Is lazy initialization used?	No	No lazy initialization	None
Initialization			in the class.	

## **Violations of Coding Standards**

Category	Checklist Item	Yes/No	Issue	Fix
Magic	Are there any	No	No magic	None
Numbers	magic numbers?		numbers present.	
Long Methods	Are there long	No	No long methods	None
	methods or		or nested loops.	

	deeply nested loops?			
Duplicated	Is there	No	No code	None
Code	duplicated		duplication.	
	code?			
Improper	Are naming	No	Constants are	Introduce
Naming	conventions		missing	constants if
Conventions	followed?		uppercase	necessary with
			naming	uppercase
			convention.	naming.

## 11. Java Code Review Checklist for Add\_Employee interface

File name	Add_Employee interface.java
class/interface	Add_Employee interface
name	

Category	Checklist	Yes/No	Issue	Fix
	Item			
Naming	Are class	Yes	None	None
Conventions	names written			
	in			
	PascalCase?			
	Are variable	Yes	None	None
	and method			
	names written			
	in camelCase?			
	Are constants	No	No constants	Consider
	written in		are used in the	adding
	uppercase		class.	constants for
	with			commonly used
	underscores?			values if
				needed.
Code Structure	Are access	Yes	None	None
	modifiers			
	used			
	correctly?			
	Are classes	Yes	None	None
	and			
	interfaces			
	separated?			

	Are packages	Yes	No package	Add package
	used		declaration in	declaration
	appropriately		the provided	if part of a
	?		code snippet.	larger
			code shippet.	project.
Method Design	Do methods	Yes	actionPerformed	None
Method Design		165	() method has	None
	have a single			
	responsibilit		the single	
	y?		responsibility	
			of handling	
			button actions.	
	Are method	Yes	None	None
	parameters			
	limited?			
	Is method	Not	No method	None
	overloading	Applicabl	overloading	
	used	е	present in this	
	properly?		class.	
Exception	Are	No	No exception	Add exception
Handling	exceptions		handling is	handling to
	handled with		used in the	handle
	try-catch		class.	potential
	blocks?			runtime
				errors during
				button
				actions or
				GUI
				initializatio
	Are specific	No	No exceptions	n. Consider
	exceptions	NO	are defined in	adding custom
	used?		this class.	exceptions if
	usear		this class.	-
				any specific
				error
				conditions
0.1	Α .	NT.	NT .	are expected.
Code	Are comments	No	No comments are	Add comments
Readability	added for		provided in the	to explain
	complex		class.	the logic for
	logic?			button
				actions and
				GUI setup.

	Is	Yes	None	None
	indentation		110110	
	consistent?			
	Are blank	Yes	None	None
	lines used to			
	separate code			
	blocks?			
	Are	Yes	Variable and	None
	meaningful		method names	
	names used		are clear and	
	for		descriptive.	
	variables,		or o	
	classes, and			
	methods?			
Performance	Are data	Not	No complex data	None
	structures	Applicabl	structures	
	chosen based	е	used.	
	on			
	performance?			
	Are costly	Not	No loops or	None
	operations	Applicabl	costly	
	minimized in	e	operations	
	loops?		present in this	
	- I I		simple class.	
	Is lazy	Not	No lazy	None
	initializatio	Applicabl	initialization	
	n used?	e	in this class.	
Memory	Are	No	Not applicable	Not
Management	unnecessary		as no resources	applicable.
_	object		requiring	
	references		cleanup are in	
	set to null?		the class.	
Security	Is user input	No	Input	Add input
	validated?		validation is	validation to
			not performed	ensure that
			on the name or	name and
			password	password
			fields.	fields are
				not empty or
				invalid.
Maintainabilit	Are there	No	The class has	None
у	long methods		no long methods	
	or deeply		or deeply	
	nested loops?		nested loops.	

Is there	No	No duplication	None
duplicated		of code in the	
code?		class.	
Are there any	No	No magic	None
magic		numbers present	
numbers?		in the class.	

Test Related Categories

Category	Checklist Item	Yes/No	Issue	Fix
Test Coverage	Are unit tests provided	No	N/A	N/A
	for all public methods and			
	critical functionalities?			
	Do unit tests cover edge	No	N/A	N/A
	cases and boundary values?			
Test Design	Are tests written	Not	N/A	N/A
	following the AAA pattern?	Applicable		
	Are individual test cases	Not	N/A	N/A
	independent?	Applicable		
	Are descriptive names used	Not	N/A	N/A
	for test methods?	Applicable		
Assertions	Are assertions used to	Not	N/A	N/A
	verify expected results?	Applicable		
	Are specific assertions	Not	N/A	N/A
	used instead of general	Applicable		
	ones?			
Boundary and	Are edge cases and	No	N/A	N/A
Edge Cases	boundary conditions			
	tested?			
	Are invalid inputs covered	No	N/A	N/A
	by tests?			
Mocking and	Are mocks or stubs used to	No	N/A	N/A
Stubbing	isolate the unit under			
	test?			
Performance	Are tests to check	No	N/A	N/A
Testing	performance for critical			
	methods provided?			
Test	Are test methods organized	Not	N/A	N/A
Maintainability	and modular?	Applicable		
	Is there a setup method	Not	N/A	N/A
	for initializing common	Applicable		
	objects?			

Housekeeping

Category	Checklist Item	Yes/No	Issue	Fix
Code Smells	Are there any	No	No code smells	None
	code smells not		identified.	

	covered by the checklist?			
Coding	Are there any	No	None	None
Standards	coding standard			
	violations not			
	covered by the			
	checklist?			
Performance	Are there any	No	The class is too	None
Inefficiencies	performance		simple to have	
	inefficiencies		performance	
	not covered by		inefficiencies.	
	the checklist?			

#### Refactoring Task 1: Replace Magic Numbers with Constants

- **Issue**: The code contains magic numbers, such as hardcoded values for button positions, window sizes, or margins, making it harder to understand and maintain.
- Refactoring Plan: Replace all magic numbers with named constants. For example, define constants like <code>BUTTON\_WIDTH</code>, <code>WINDOW\_MARGIN</code>, or <code>FONT\_SIZE</code> at the beginning of the class or in a dedicated constants section.
  - o BUTTON\_WIDTH = 100;
  - o WINDOW\_MARGIN = 10;
  - FONT SIZE = 14;
- **Benefit**: Improves code readability by providing meaningful names for values. Makes it easier to update values in one place without the risk of inconsistent changes throughout the code.

#### Refactoring Task 2: Consolidate Redundant Method Calls

- **Issue**: The same method is being called multiple times with the same parameters, causing unnecessary repetition and reducing code efficiency.
- **Refactoring Plan**: Consolidate redundant method calls by storing the result in a local variable, then reusing the variable instead of calling the method again.
  - For example, if the method getEmployeeInfo() is called multiple times with the same input, store the result in a variable:
- Benefit: Reduces unnecessary method calls, making the code more efficient. It also reduces the risk of calling the method multiple times when it might return different results.

#### Refactoring Task 3: Simplify Conditional Expressions

• **Issue**: Complex and nested conditional expressions make the code difficult to read and understand.

- **Refactoring Plan:** Simplify complex if-else or ternary operations by breaking them into smaller, easier-to-read conditions, or using return statements early in the method to avoid deep nesting.
- **Benefit**: Improves code readability and simplifies control flow, making it easier to maintain and modify. Reduces the likelihood of introducing errors due to complex nesting.

#### 12. Java Code Review Checklist for Admin interface

File name	Admin interface. java
class/interface	Admininterface
name	

Category	Checklist	Yes/No	Issue	Fix
	Item			
Naming	Are class	Yes	None	None
Conventions	names written			
	in			
	PascalCase?			
	Are variable	Yes	None	None
	and method			
	names written			
	in camelCase?			
	Are constants	No	No constants	Consider
	written in		are used in	adding
	uppercase		the class.	constants for
	with			commonly used
	underscores?			values if
				needed.
Code Structure	Are access	Yes	None	None
	modifiers			
	used			
	correctly?			
	Are classes	Yes	None	None
	and			
	interfaces			
	separated?			
	Are packages	Yes	No package	Add package
	used		declaration in	declaration if
	appropriately		the provided	part of a
	?		code snippet.	larger
				project.

Method Design	Do methods	Yes	actionPerforme	Split the
Method Design	have a single	168	d() method	actionPerforme
			handles	d() method
	responsibilit			"
	у?		multiple	into smaller
			button	methods to
			actions. It	handle
			could be split	individual
			for better	button
			clarity.	actions.
	Are method	Yes	None	None
	parameters			
	limited?			
	Is method	Not	No method	None
	overloading	Applicab	overloading	
	used	le	present in	
	properly?		this class.	
Exception	Are	No	No exception	Add exception
Handling	exceptions		handling is	handling to
	handled with		used in the	handle
	try-catch		class.	potential
	blocks?		C1435.	runtime errors
	DIOCKS:			
				during button
				actions or GUI
				initialization
		NT.	AT	
	Are specific	No	No exceptions	Consider
	exceptions		are defined in	adding custom
	used?		this class.	exceptions if
				any specific
				error
				conditions are
				expected.
Code	Are comments	No	No comments	Add comments
Readability	added for		are provided	to explain the
	complex		in the class.	logic for
	logic?			button actions
				and GUI setup.
	Is	Yes	None	None
	indentation			
	consistent?			
	Are blank	Yes	None	None
	lines used to			
	separate code			
	blocks?			
	DIOCKS:	<u> </u>		

	Are	Yes	Variable and	None
	meaningful		method names	
	names used		are clear and	
	for		descriptive.	
	variables,			
	classes, and			
	methods?			
Performance	Are data	Not	No complex	None
	structures	Applicab	data	
	chosen based	1e	structures	
	on		used.	
	performance?			
	Are costly	Not	No loops or	None
	operations	Applicab	costly	
	minimized in	1e	operations	
	loops?		present in	
			this simple	
			class.	
	Is lazy	Not	No lazy	None
	initializatio	Applicab	initialization	
	n used?	1e	in this class.	
Memory	Are	No	Not applicable	Not
Management	unnecessary		as no	applicable.
	object		resources	
	references		requiring	
	set to null?		cleanup are in	
			the class.	
Security	Is user input	No	Input	Add input
•	validated?		validation is	validation to
			not performed	ensure that
			on any fields	text fields
			in the GUI.	(e. g.,
				username,
				password) are
				not empty and
				follow
				required
				formats.
Maintainabilit	Are there	Yes	The	Break the
у	long methods		actionPerforme	actionPerforme
	or deeply		d() method is	d() method
	nested loops?		too long and	into smaller,
			handles too	more
			many actions.	
		<u> </u>	many actions.	<u> </u>

			manageable methods.
Is there	No	No duplication	None
duplicated		of code in the	
code?		class.	
Are there any	No	No magic	None
magic		numbers	
numbers?		present in the	
		class.	
Tes	t Related G	Ouries	

Category	Checklist Item	Yes/No	Issue	Fix
Test Coverage	Are unit tests provided for	No	N/A	N/A
	all public methods and			
	critical functionalities?			
	Do unit tests cover edge cases	No	N/A	N/A
	and boundary values?			
Test Design	Are tests written following	Not	N/A	N/A
	the AAA pattern?	Applicable		
	Are individual test cases	Not	N/A	N/A
	independent?	Applicable		
	Are descriptive names used for	Not	N/A	N/A
	test methods?	Applicable		
Assertions	Are assertions used to verify	Not	N/A	N/A
	expected results?	Applicable		
	Are specific assertions used	Not	N/A	N/A
	instead of general ones?	Applicable		
Boundary and	Are edge cases and boundary	No	N/A	N/A
Edge Cases	conditions tested?			
	Are invalid inputs covered by	No	N/A	N/A
	tests?			
Mocking and	Are mocks or stubs used to	No	N/A	N/A
Stubbing	isolate the unit under test?			
Performance	Are tests to check performance	No	N/A	N/A
Testing	for critical methods provided?			
Test	Are test methods organized and	Not	N/A	N/A
Maintainability	modular?	Applicable		
	Is there a setup method for	Not	N/A	N/A
	initializing common objects?	Applicable	_	

### Housekeeping

Category	Checklist Item	Yes/No	Issue	Fix
Code Smells	Are there any code	No	No code smells	None
	smells not covered		identified.	
	by the checklist?			

Coding	Are there any	No	None	None
Standards	coding standard			
	violations not			
	covered by the			
	checklist?			
Performance	Are there any	No	The class is simple	None
Inefficiencies	performance		and does not have	
	inefficiencies not		performance	
	covered by the		inefficiencies.	
	checklist?			

#### Refactoring Task 1: Extract Duplicate Code into a Helper Method

- **Issue**: The code contains duplicated logic, such as multiple places where the same block of code is repeated, making it harder to maintain and modify.
- **Refactoring Plan**: Identify the duplicated code, extract it into a helper method, and then call this method wherever the logic is needed.
- **Benefit**: Reduces redundancy, making the code more maintainable and easier to modify. Changes to the logic can be made in one place, improving consistency across the codebase.

#### Refactoring Task 2: Use StringBuilder for String Concatenation

- **Issue**: The code uses the + operator to concatenate strings in loops or frequently called methods, which can be inefficient and lead to performance issues.
- Refactoring Plan: Replace string concatenation with a StringBuilder to improve performance, especially in loops or methods that are executed frequently.
- **Benefit**: Improves performance by reducing the overhead of creating new string objects during concatenation. It also enhances readability when dealing with complex string operations.

#### Refactoring Task 3: Break Down Large Methods into Smaller Methods

- **Issue**: A method is too long or complex, making it difficult to read, test, and maintain.
- **Refactoring Plan**: Break the large method into smaller, more manageable methods, each responsible for a specific part of the functionality.
- **Benefit**: Makes the code more modular, easier to test, and improves readability. It also helps in maintaining single responsibility for each method.

						•	
A	$\sim$	$\sim$	$\sim$	_	$\sim$	11	, ·
$\boldsymbol{H}$			_			ΙX	
, v	2	~	v		u	17	٠.

**Code Smells** 

Code Smell	Description	Example	Suggested Solution
Long Methods	Methods that perform multiple tasks, making them hard to understand and maintain.	The updateInventory method combines inventory updating and file I/O logic.	Split updateInventory into smaller methods: one for inventory updating and another for file operations.
Nested Loops	Deeply nested loops reduce readability and increase complexity.	Nested looping over transactionItem and databaseItem.	Use a HashMap to cache databaseItem by itemID to minimize nested loops.
Duplicate Code	Repeated code for file handling and reading/writing inventory data.	File reading and writing logic appears in multiple places.	Extract file handling logic into reusable helper methods.
Magic Numbers/Strings	Hardcoded values or strings used without explanation.	Hardcoded file paths like "inventory.txt" and string keys like "ItemID".	Define constants for file paths and string keys (e.g., FILE_PATH_INVENTORY, KEY_ITEM_ID).
Primitive Obsession	Overuse of primitive types instead of meaningful abstractions.	Using strings to represent itemID or transaction status instead of enums or classes.	Use an enum for transaction types and consider using a class for inventory items.

### **Performance Inefficiencies**

Issue	Description	Example	Suggested Solution	
Repeated	Scanning the entire	Iterating through the	Use a	
Object	databaseItem list	entire list for each	HashMap <integer,< td=""></integer,<>	
Lookups	for every	transaction to find the	Item> to store	
	transactionItem.	matching item.	databaseItem for	

			constant-time
			lookups.
Redundant	Performing the	Repeated calls to	Store itemID in a local
Calculations	same calculation	getItemID() inside	variable before the
	repeatedly in a	nested loops.	loop to minimize
	loop.		method calls.
Inefficient	Using string	Using result += "Item:	Use StringBuilder for
String Ops	concatenation	" + itemName in a	concatenating strings
	inside loops.	loop.	inside loops to
			improve performance.
Excessive	Printing to the	Using	Replace
Logging	console	System.out.println for	System.out.println
	excessively instead	error reporting in	with a proper logging
	of structured	catch blocks.	framework like
	logging.		java.util.logging or
			Log4j.

### Housekeeping

Category	Issue	Fix
Code Smells	Duplicate file handling code.	Extract file reading/writing logic
		into dedicated helper methods.
	Combined logic in	Separate updateInventory into
	updateInventory that violates	modular methods for inventory
	single responsibility	processing and file updates.
	principle.	
Coding	Hardcoded strings like file	Define constants for all
Standards	paths and key names.	commonly used strings.
	Inconsistent error handling	Replace direct System.out.println
	(direct printing in catch	calls with proper logging using a
	blocks).	logging framework.
Performance	Repeated list traversal for	Cache databaseltem in a
Inefficiencies	inventory lookup.	HashMap for efficient lookups.
	Direct string concatenation	Use StringBuilder for string
	inside loops.	manipulations inside loops.

## 13. Java Code Review Checklist for Cashier interface

File name	Casier interface. java
class/interface	CashierInterface
name	

Category	Checklist Item	Yes/No	Issue	Fix
Naming Conventions	Are class names written in PascalCase?	Yes	None	None
	Are variable and method names written in camelCase?	Yes	None	None
	Are constants written in uppercase with underscores ?	No	No constants are used in the class.	Add constants for commonly reused values (e.g., button heights).
Code Structure	Are access modifiers used correctly?	Yes	None	None
	Are classes and interfaces separated?	Yes	None	None
	Are packages used appropriate 1y?	No	No package declaration in the class.	Add package declaration if it is part of a larger project.
Method Design	Do methods have a single	No	The actionPerforme d() method handles	Split actionPerformed( ) into smaller methods for

responsibil multiple individua	11
ity? button button	
actions.	3 0 1 0 11 5 1
Are method Yes None None	
parameters	
limited?	
Is method Not No method None	
overloading Applica overloading	
used ble present.	
properly?	
Exception Are No No exception Add excep	otion
Handling exceptions handling for handling	
handled invalid input. parsing p	
with try-	
catch	
blocks?	
Are No No specific Add speci	ific
specific exceptions are exception	
exceptions defined. handling	
used? expected	
Code Are No No comments Add comme	
Readability comments explaining transact	
added for business restorat:	
complex logic. logout be	
logic?	
Is Yes None None	
indentation	
consistent?	
Are blank Yes None None	
lines used	
to separate	
code	
blocks?	
Are Yes Variable names None	
meaningful are meaningful	
names used and	
for descriptive.	
variables,	
classes,	
and	
methods?	
Performance         Are data         Not         No complex         None	
structures Applica data	
chosen ble	

	based on		structures are	
	performance		used.	
	?			
	Are costly	Not	No costly	None
	operations	Applica	operations in	
	minimized	ble	loops.	
	in loops?			
	Is lazy	Not	No lazy	None
	initializat	Applica	initialization	
	ion used?	ble	is used.	
Memory	Are	No	Unused	Set unused
Management	unnecessary		resources are	Transaction_Inte
	object		not explicitly	rface objects to
	references		cleared.	null after
	set to			disposal.
	nu11?			
Security	Is user	No	Phone number	Validate user
	input		input is not	inputs to ensure
	validated?		validated.	correctness and
				prevent crashes.
Maintainability	Are there	Yes	The	Refactor
	long		actionPerforme	actionPerformed(
	methods or		d() method is	) for better
	deeply		long.	maintainability.
	nested			
	loops?			
	Is there	Yes	Repeated	Extract common
	duplicated		disposal logic	disposal logic
	code?		in	into a helper
			actionPerforme	method.
			d().	
	Are there	Yes	Button height	Replace with
	any magic		and positions	constants or
	numbers?		are hardcoded.	calculations
				based on screen
				size.

Test Related Categories

Category	y Checklist Item		Issue	Fix
Test Coverage	Are unit tests provided for	No	N/A	N/A
	all public methods and			
	critical functionalities?			
	Do unit tests cover edge	No	N/A	N/A
	cases and boundary values?			

Test Design	Are tests written following	Not		N/A	N/A
	the AAA pattern?	App1	icable		
	Are individual test cases	Not		N/A	N/A
	independent?	App1	icable		
	Are descriptive names used	Not		N/A	N/A
	for test methods?	App1	icable		
Assertions	Are assertions used to	Not		N/A	N/A
	verify expected results?	App1	icable		
	Are specific assertions used	Not		N/A	N/A
	instead of general ones?	App1	icable		
Boundary and	Are edge cases and boundary	No		N/A	N/A
Edge Cases	conditions tested?				
	Are invalid inputs covered	No		N/A	N/A
	by tests?				
Mocking and	Are mocks or stubs used to	No		N/A	N/A
Stubbing	isolate the unit under test?				
Performance	Are tests to check	No		N/A	N/A
Testing	performance for critical				
	methods provided?				
Test	Are test methods organized	Not		N/A	N/A
Maintainability	and modular?	App1	icable		
	Is there a setup method for	Not		N/A	N/A
	initializing common objects?	App1	icable		
	Housekeeping				
Category	Checklist Item	Y	es/No	Issue	Fix
Code Smells	Are there any code smells no	ot N	О	None	None
	covered by the checklist?				
Coding Standards	Are there any coding standar	rd N	О	None	None
	violations not covered by the	ne			
	checklist?				
Performance	Are there any performance	N	О	None	None
Inefficiencies	inefficiencies not covered b	ру			
	the checklist?				

## Refactoring Task 1: Extract Duplicate Code into a Helper Method

- **Issue:** The actionPerformed() method contains repeated logic for initializing and disposing of the Transaction\_Interface object.
- Refactoring Plan:
  - 1. Create a helper method (e.g., openTransaction (String operation)) to handle the repeated initialization and disposal logic.
  - 2. Replace the repetitive code in the actionPerformed() method with calls to this helper method.

• **Benefit**: Reduces redundancy, making the code more maintainable. Any changes to the initialization logic can now be made in a single location.

#### Refactoring Task 2: Replace Magic Numbers with Constants

- **Issue**: Hardcoded values for button dimensions and positions are scattered throughout the class, making it harder to modify or understand.
- Refactoring Plan:
  - 1. Define constants (e.g., BUTTON\_HEIGHT, BUTTON\_WIDTH, BUTTON VERTICAL SPACING) to represent these magic numbers.
  - 2. Replace all instances of these hardcoded values with the defined constants.
- **Benefit:** Improves readability and maintainability by giving meaningful names to numeric values. Changes to button dimensions or layout can now be made by updating the constants in one place.

#### Refactoring Task 3: Break Down Large Methods into Smaller Methods

- **Issue**: The actionPerformed() method is long and handles multiple unrelated button actions, violating the single-responsibility principle.
- Refactoring Plan:
  - Extract the logic for each button action into separate methods (e.g., handleSaleAction(), handleRentalAction(), handleReturnAction(), handleLogOutAction()).
  - 2. Update actionPerformed() to delegate tasks to these smaller methods based on the event source.
- **Benefit**: Enhances readability, makes the code more modular, and simplifies testing. Each button action can now be tested independently.

## Appendix:

#### Code Smells

Code Smell	Description	Example	Suggested Solution
Long Methods	Methods that	The	Split updateInventory into
	perform	updateInventory	smaller methods: one for
	multiple tasks,	method	inventory updating and
	making them	combines	another for file
	hard to	inventory	operations.
	understand and	updating and file	
	maintain.	I/O logic.	

Nested Loops	Deeply nested	Nested looping	Use a HashMap to cache
	loops reduce	over	databaseItem by itemID
	readability and	transactionItem	to minimize nested loops.
	increase	and	
	complexity.	databaseltem.	
<b>Duplicate Code</b>	Repeated code	File reading and	Extract file handling logic
	for file handling	writing logic	into reusable helper
	and	appears in	methods.
	reading/writing	multiple places.	
	inventory data.		
Magic	Hardcoded	Hardcoded file	Define constants for file
Numbers/Strings	values or strings	paths like	paths and string keys
	used without	"inventory.txt"	(e.g.,
	explanation.	and string keys	FILE_PATH_INVENTORY,
		like "ItemID".	KEY_ITEM_ID).
Primitive	Overuse of	Using strings to	Use an enum for
Obsession	primitive types	represent itemID	transaction types and
	instead of	or transaction	consider using a class for
	meaningful	status instead of	inventory items.
	abstractions.	enums or	
		classes.	

#### **Performance Inefficiencies**

Issue	Description	Example	Suggested Solution
Repeated	Scanning the entire	Iterating through the	Use a
Object	databaseItem list	entire list for each	HashMap <integer,< th=""></integer,<>
Lookups	for every	transaction to find the	Item> to store
	transactionItem.	matching item.	databaseItem for
			constant-time
			lookups.
Redundant	Performing the	Repeated calls to	Store itemID in a local
Calculations	same calculation	getItemID() inside	variable before the
	repeatedly in a	nested loops.	loop to minimize
	loop.		method calls.

Inefficient	Using string	Using result += "Item:	Use StringBuilder for
String Ops	concatenation	" + itemName in a	concatenating strings
	inside loops.	loop.	inside loops to
			improve performance.
<u> </u>	B		5 1
Excessive	Printing to the	Using	Replace
Logging	console	System.out.println for	System.out.println
	excessively instead	error reporting in	with a proper logging
	of structured	catch blocks.	framework like
	logging.		java.util.logging or
			Log4j.

## Housekeeping

Category	Issue	Fix
Code Smells	Duplicate file handling code.	Extract file reading/writing logic
		into dedicated helper methods.
	Combined logic in	Separate updateInventory into
	updateInventory that violates	modular methods for inventory
	single responsibility	processing and file updates.
	principle.	
Coding	Hardcoded strings like file	Define constants for all
Standards	paths and key names.	commonly used strings.
	Inconsistent error handling	Replace direct System.out.println
	(direct printing in catch	calls with proper logging using a
	blocks).	logging framework.
Performance	Repeated list traversal for	Cache databaseItem in a
Inefficiencies	inventory lookup.	HashMap for efficient lookups.
	Direct string concatenation	Use StringBuilder for string
	inside loops.	manipulations inside loops.

## 14. Java Code Review Checklist for Enter\_Item interface

File name	EnterItem_interface.java
class/interface	EnterItem_Interface
name	

Category	Checklist	Yes/No	Issue	Fix
	Item			
Naming Conventions	Are class names written in PascalCas e?	Yes	None	None
	Are variable and method names written in camelCase ?	Yes	None	None
	Are constants written in uppercase with underscor es?	No	No constants are used in the class.	Add constants for GUI dimensions and reusable values.
Code Structure	Are access modifiers used correctly ?	Yes	None	None
	Are classes and interface s separated?	Yes	None	None
	Are packages used appropria tely?	No	No package declaration in the class.	Add package declaration if it's part of a larger project.

Method Design	Do	No	The	Split
LISTING DOSIGII	methods	110	actionPerform	actionPerformed()
	have a		ed() method	into smaller
	single		handles	methods for
	responsib		multiple	individual
	ility?		actions.	actions.
	Are	Yes	None	None
	method	168	None	None
	parameter			
	S			
	limited?			
	Is method	Not	No method	None
	overloadi	Applica	overloading	None
	ng used	ble		
	_	bre	present.	
Exception Handling	properly? Are	No	No exception	Add oxcontion
Exception nanding		NO	handling for	Add exception handling for
	exception s handled		invalid	_
				getItemID() and
	with try- catch		input.	getAmount().
	blocks?			
	-	No	General	C-+-1
	Are	NO		Catch specific
	specific		parsing	exceptions like
	exception		errors not	NumberFormatExcep
O 1 D 11:1:4	s used?	NT.	handled.	tion.
Code Readability	Are	No	No comments	Add comments for
	comments		explaining	item
	added for		business	addition/removal
	complex		logic.	logic and
	logic?	T/	NT.	updateTextArea.
	Is	Yes	None	None
	indentati			
	on			
	consisten			
	t?	V	NT .	N.
	Are blank	Yes	None	None
	lines			
	used to			
	separate			
	code			
	blocks?	7.7	77 • 1 7	) T
	Are	Yes	Variable and	None
	meaningfu		method names	
	1 names		are	

	used for		meaningful	
	variables			
	variables		and	
	,		descriptive.	
	classes,			
	and			
	methods?			
Performance	Are data	Not	No complex	None
	structure	Applica	data	
	s chosen	ble	structures	
	based on		are used.	
	performan		are asea.	
	ce?			
	-	NT.	C. ·	II C D :11
	Are	No	String	Use StringBuilder
	costly		concatenation	for string
	operation		in	concatenation in
	S		updateTextAre	loops.
	minimized		a() could be	
	in loops?		optimized.	
	Is lazy	Not	No lazy	None
	initializ	Applica	initializatio	
	ation	ble	n is used.	
	used?	DIC	n is asca.	
Memory Management	Are	No	Unused	Set unused
	unnecessa		resources not	resources to null
	ry object		cleared	post-disposal if
	reference		explicitly.	needed.
			expireitly.	needed.
	s set to			
a	nu11?	NT.	37 1.1	77 1 • 1 .
Security	Is user	No	I No wolidotion	
		110	No validation	Validate user
	input	110	for itemID	inputs to ensure
	input validated	110	for itemID and amount	
	input		for itemID	inputs to ensure
Maintainability	input validated	Yes	for itemID and amount	inputs to ensure numeric values
Maintainability	input validated ?		for itemID and amount fields.	inputs to ensure numeric values are entered.
Maintainability	input validated ? Are there		for itemID and amount fields. actionPerform	inputs to ensure numeric values are entered.
Maintainability	input validated ? Are there long methods		for itemID and amount fields. actionPerform ed() is long and handles	inputs to ensure numeric values are entered. Refactor actionPerformed()
Maintainability	input validated ? Are there long methods or deeply		for itemID and amount fields. actionPerform ed() is long and handles multiple	inputs to ensure numeric values are entered.  Refactor actionPerformed() into smaller, more cohesive
Maintainability	input validated ? Are there long methods or deeply nested		for itemID and amount fields. actionPerform ed() is long and handles	inputs to ensure numeric values are entered.  Refactor actionPerformed() into smaller,
Maintainability	input validated ? Are there long methods or deeply nested loops?		for itemID and amount fields. actionPerform ed() is long and handles multiple concerns.	inputs to ensure numeric values are entered.  Refactor actionPerformed() into smaller, more cohesive methods.
Maintainability	input validated ? Are there long methods or deeply nested loops? Is there	Yes	for itemID and amount fields. actionPerform ed() is long and handles multiple concerns.  Repeated	inputs to ensure numeric values are entered.  Refactor actionPerformed() into smaller, more cohesive methods.  Extract common
Maintainability	input validated ? Are there long methods or deeply nested loops? Is there duplicate	Yes	for itemID and amount fields. actionPerform ed() is long and handles multiple concerns.  Repeated disposal	inputs to ensure numeric values are entered.  Refactor actionPerformed() into smaller, more cohesive methods.  Extract common disposal logic
Maintainability	input validated ? Are there long methods or deeply nested loops? Is there	Yes	for itemID and amount fields. actionPerform ed() is long and handles multiple concerns.  Repeated disposal logic in	inputs to ensure numeric values are entered.  Refactor actionPerformed() into smaller, more cohesive methods.  Extract common disposal logic into a helper
Maintainability	input validated ? Are there long methods or deeply nested loops? Is there duplicate	Yes	for itemID and amount fields. actionPerform ed() is long and handles multiple concerns.  Repeated disposal	inputs to ensure numeric values are entered.  Refactor actionPerformed() into smaller, more cohesive methods.  Extract common disposal logic

	Are there	Yes	Button a	nd	Ren1	lace wit	:h
	any magic	103	text fie			onstants or	
	numbers?		positions an			layout manager	
	Humbers:		hardcode		Tayout managers		igcis.
	Test	Related Ca	l	u.			
Category		cklist Ite		Yes	:/No	Issu	e Fix
Test Coverage	Are unit te	ests provi	ded for	No		N/A	N/A
_		11 public methods and					
	critical fu						
	Do unit tes	sts cover	edge	No		N/A	N/A
	cases and h						
Test Design	Are tests v			Not		N/A	N/A
_	the AAA pat	tern?	_	Appli	cab1e	)	
	Are individ	dual test	cases	Not		N/A	N/A
	independent			Appli	cab1e	·	
	Are descrip		s used	Not		N/A	N/A
	for test me			Appli	cab1e	,	
Assertions	Are asserti	ions used	to	Not		N/A	N/A
	verify expe	ected resu	lts?	Appli	cab1e	,	
	Are specifi	c asserti	ons used	Not		N/A	N/A
	instead of	general o	nes?	Appli	cab1e	,	
Boundary and	Are edge ca	ses and b	oundary	No		N/A	N/A
Edge Cases		conditions tested?					
	Are invalid	Are invalid inputs covered No			N/A	N/A	
	by tests?						
Mocking and	Are mocks o	or stubs u	sed to	No		N/A	N/A
Stubbing	isolate the	e unit und	er test?				
Performance	Are tests t	to check		No		N/A	N/A
Testing	performance	e for crit	ical				
	methods pro	ovided?					
Test	Are test me	ethods org	anized	Not		N/A	N/A
Maintainability	and modular	?		Appli	cab1e	)	
	Is there a	setup met	hod for	Not		N/A	N/A
	initializir	ng common	objects?	Appli	cab1e	)	
		Housekeep	ing	1			
Category	C	hecklist ]	[tem	Ye	s/No	Issue	Fix
Code Smells	Are there	any code	smells no	ot No		None	None
	covered b	y the chec	cklist?				
Coding Standards	Are there	any codin	ng standar	d No		None	None
	violation	violations not covered by the					
	checklist	?					
Performance		any perfo		No		None	None
Inefficiencies	inefficie	inefficiencies not covered by					
	the check	list?					

#### Refactoring Task 1: Extract Repeated Code into a Helper Method

- **Issue**: The actionPerformed() method contains repeated logic for disposing of the interface after completing an action.
- Refactoring Plan:
  - 1. Create a helper method, disposeInterface(), to handle the repeated logic for setting visibility to false and disposing of the interface.
  - 2. Replace all instances of this repeated logic in the actionPerformed() method with calls to disposeInterface().
- **Benefit**: Reduces redundancy, making the code more maintainable. Changes to the disposal logic can now be made in a single location.

#### Refactoring Task 2: Replace Magic Numbers with Constants

- **Issue**: Hardcoded values for button dimensions, positions, and component spacing are scattered throughout the class.
- Refactoring Plan:
  - 1. Define constants (e.g., BUTTON\_WIDTH, BUTTON\_HEIGHT, TEXTFIELD\_WIDTH, TEXTFIELD HEIGHT) to represent these magic numbers.
  - 2. Replace all instances of these hardcoded values in the layout setup with the defined constants.
- **Benefit**: Improves readability and maintainability by giving meaningful names to numeric values. Changes to layout dimensions can now be made in one place.

#### Refactoring Task 3: Break Down Large Methods into Smaller Methods

- **Issue:** The actionPerformed() method is lengthy and handles multiple unrelated actions, violating the single-responsibility principle.
- Refactoring Plan:
  - Extract the logic for handling different button actions into separate methods (e.g., handleEnterButtonAction(), handleExitButtonAction()).
  - 2. Update the actionPerformed() method to delegate tasks to these smaller methods based on the event source.
- Benefit: Enhances readability, modularity, and testability. Each button's functionality can now be tested and modified independently.

# Appendix:

## **Code Smells**

Code Smell	Description	Example	Suggested Solution
Long Methods	Methods that perform multiple tasks, making them hard to understand and maintain.	The updateInventory method combines inventory updating and file I/O logic.	Split updateInventory into smaller methods: one for inventory updating and another for file operations.
Nested Loops	Deeply nested loops reduce readability and increase complexity.	Nested looping over transactionItem and databaseItem.	Use a HashMap to cache databaseItem by itemID to minimize nested loops.
Duplicate Code	Repeated code for file handling and reading/writing inventory data.	File reading and writing logic appears in multiple places.	Extract file handling logic into reusable helper methods.
Magic Numbers/Strings	Hardcoded values or strings used without explanation.	Hardcoded file paths like "inventory.txt" and string keys like "ItemID".	Define constants for file paths and string keys (e.g., FILE_PATH_INVENTORY, KEY_ITEM_ID).
Primitive Obsession	Overuse of primitive types instead of meaningful abstractions.	Using strings to represent itemID or transaction status instead of enums or classes.	Use an enum for transaction types and consider using a class for inventory items.

#### **Performance Inefficiencies**

Issue	Description	Example	Suggested Solution

Repeated	Scanning the entire	Iterating through the	Use a
Object	databaseItem list	entire list for each	HashMap <integer,< th=""></integer,<>
Lookups	for every	transaction to find the	Item> to store
	transactionItem.	matching item.	databaseItem for
			constant-time
			lookups.
Redundant	Performing the	Repeated calls to	Store itemID in a local
Calculations	same calculation	getItemID() inside	variable before the
	repeatedly in a	nested loops.	loop to minimize
	loop.		method calls.
Inefficient	Using string	Using result += "Item:	Use StringBuilder for
String Ops	concatenation	" + itemName in a	concatenating strings
	inside loops.	loop.	inside loops to
			improve performance.
Excessive	Printing to the	Using	Replace
Logging	console	System.out.println for	System.out.println
	excessively instead	error reporting in	with a proper logging
	of structured	catch blocks.	framework like
	logging.		java.util.logging or
			Log4j.

## Housekeeping

Category	Issue	Fix
Code Smells	Duplicate file handling code.	Extract file reading/writing logic into dedicated helper methods.
	Combined logic in updateInventory that violates single responsibility principle.	Separate updateInventory into modular methods for inventory processing and file updates.
Coding Standards	Hardcoded strings like file paths and key names.	Define constants for all commonly used strings.
	Inconsistent error handling (direct printing in catch blocks).	Replace direct System.out.println calls with proper logging using a logging framework.

Performance	Repeated list traversal for	Cache databaseltem in a	
Inefficiencies inventory lookup.		HashMap for efficient lookups.	
Direct string concatenation		Use StringBuilder for string	
inside loops.		manipulations inside loops.	
	·		

# 15. Java Code Review Checklist for Login interface

File name	Login_interface. java
class/interface	Login_Interface
name	

Category	Checklist	Yes/No	Issue	Fix
	Item			
Naming	Are class	Yes	None	None
Conventions	names			
	written in			
	PascalCase			
	?			
	Are	Yes	None	None
	variable			
	and method			
	names			
	written in			
	camelCase?			
	Are	No	No constants	Consider adding
	constants		are used in	constants for GUI
	written in		the class.	layout (e.g.,
	uppercase			button sizes,
	with			window size).
	underscore			
	s?			
Code Structure	Are access	Yes	None	None
	modifiers			
	used			
	correctly?			
	Are	Yes	None	None
	classes			
	and			
	interfaces			
	separated?			

	Are	No	The class is	Add a package
	packages		not part of	declaration if it's
	used		any package.	part of a larger
	appropriat			project.
	ely?			
Method Design	Do methods	No	The	Split
	have a		actionPerfor	actionPerformed()
	single		med() method	into smaller
	responsibi		handles	methods for login
	lity?		login,	and UI updates.
			validation,	
			and UI	
			updates.	
	Are method	Yes	None	None
	parameters			
	limited?			
	Is method	Not	No method	None
	overloadin	Applica	overloading	
	g used	ble	present.	
	properly?			
Exception	Are	No	No exception	Add exception
Handling	exceptions		handling for	handling for
	handled		invalid	invalid username or
	with try-		input or	password format.
	catch		login	
	blocks?		errors.	
	Are	No	General	Use specific
	specific		login	exceptions like
	exceptions		failure is	InvalidCredentialsE
	used?		handled	xception to improve
			without	clarity.
			specific	
Code	Ano	No	exceptions. The code	Add comments
	Are	No	lne code lacks	
Readability	comments added for		comments	explaining the login flow and
	complex			error handling.
	logic?		explaining key	error namarring.
	TOSIC:		sections,	
			especially	
			for login	
			validation.	
	Is	Yes	None	None
	indentatio	103	NOHE	HOHE
	muentatio	<u> </u>		

	10			
	n			
	consistent			
	?	**	77	17
	Are blank	Yes	None	None
	lines used			
	to			
	separate			
	code			
	blocks?			
	Are	Yes	Variable and	None
	meaningful		method names	
	names used		are	
	for		meaningful.	
	variables,		_	
	classes,			
	and			
	methods?			
Performance	Are data	Not	No data	None
	structures	Applica	structures	
	chosen	ble	are used for	
	based on	210	complex	
	performanc		operations.	
	e?		operations.	
	Are costly	Not	No loops	None
	operations	Applica	present.	None
	minimized	ble	present.	
	in loops?	DIE		
	Is lazy	Not	No lazy	None
	initializa	Applica	initializati	None
	tion used?	ble	on used.	
Memory	Are	No	There is no	Consider setting
-	unnecessar	NO	explicit	unused references
Management				
	y object		clearing of	to null after they
	references		references.	are no longer
	set to			needed.
C	null?	NT .	T	A 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Security	Is user	No	Input	Add validation to
	input		validation	check for non-empty
	validated?		for username	inputs and invalid
			and password	characters.
			fields is	
			missing.	
Maintainability	Are there	Yes	actionPerfor	Refactor the
	long		med() method	actionPerformed()

methods or		is long and	method into
deeply		performs	smaller, more
nested		multiple	cohesive methods.
loops?		actions.	
Is there	Yes	The login	Extract login logic
duplicated		logic is	into a separate
code?		repeated for	method to avoid
		checking	repetition.
		user	
		credentials.	
Are there	Yes	The window	Replace with
any magic		dimensions	constants for GUI
numbers?		and button	elements (e.g.,
		positions	window size, button
		are	positions).
		hardcoded.	

Test Related Categories

Category	Checklist Item	Yes/No	Issue	Fix
Test Coverage	Are unit tests provided for	No	N/A	N/A
	all public methods and			
	critical functionalities?			
	Do unit tests cover edge	No	N/A	N/A
	cases and boundary values?			
Test Design	Are tests written following	Not	N/A	N/A
	the AAA pattern?	Applicable		
	Are individual test cases	Not	N/A	N/A
	independent?	Applicable		
	Are descriptive names used	Not	N/A	N/A
	for test methods?	Applicable		
Assertions	Are assertions used to	Not	N/A	N/A
	verify expected results?	Applicable		
	Are specific assertions used	Not	N/A	N/A
	instead of general ones?	Applicable		
Boundary and	Are edge cases and boundary	No	N/A	N/A
Edge Cases	conditions tested?			
	Are invalid inputs covered	No	N/A	N/A
	by tests?			
Mocking and	Are mocks or stubs used to	No	N/A	N/A
Stubbing	isolate the unit under test?			
Performance	Are tests to check	No	N/A	N/A
Testing	performance for critical			
	methods provided?			
Test	Are test methods organized	Not	N/A	N/A
Maintainability	and modular?	Applicable		

	Is there a setup method for	Not	N/A	N/A	
	initializing common objects?	Applicable	)		
	Housekeeping				
Category	Checklist Item	Yes/No	Issue	Fix	
Code Smells	Are there any code smells no	t No	None	None	
	covered by the checklist?				
Coding Standards	Are there any coding standar	d No	None	None	
	violations not covered by the	е			
	checklist?				
Performance	Are there any performance	No	None	None	
Inefficiencies	inefficiencies not covered b				
	the checklist?	the checklist?			

#### Refactoring Task 1: Extract Repeated Code into a Helper Method

- **Issue**: The logic for disposing of the interface after completing an action is repeated when navigating to the Cashier Interface and Admin Interface.
- Refactoring Plan:
  - 1. Create a helper method, disposeInterface(), that handles the repeated logic of setting visibility to false and disposing of the interface.
  - 2. Replace all instances of this logic in the actionPerformed() method with calls to disposeInterface().
- **Benefit**: Reduces redundancy, making the code more maintainable. Changes to the disposal logic can now be made in a single location, improving clarity and reducing the chances of errors when updating the code.

#### Refactoring Task 2: Replace Magic Numbers with Constants

- **Issue**: Hardcoded values for window size, button positions, and component dimensions (e.g., button width, height, and position) are scattered throughout the code.
- Refactoring Plan:
  - 1. Define constants at the top of the class to represent these magic numbers (e.g., window\_width, window\_Height, Button\_width, Button Height, etc.).

- 2. Replace all instances of these hardcoded values in the layout setup with the defined constants.
- **Benefit**: Improves readability and maintainability by giving meaningful names to numeric values. When layout changes are needed, they can now be done in one place, reducing potential for errors and improving scalability.

#### Refactoring Task 3: Break Down Large Method into Smaller Methods

- Issue: The actionPerformed() method is lengthy and handles multiple responsibilities (handling login, displaying error messages, and managing interface transitions).
- Refactoring Plan:
  - 1. Extract the logic for handling the login action, error display, and interface transitions into separate methods (e.g., handleLoginAction(), showErrorMessage(), navigateToCashier(), navigateToAdmin()).
  - 2. Update the actionPerformed() method to delegate tasks to these smaller methods based on the event source.
- **Benefit**: Enhances code readability and modularity. Each action (login, error handling, UI navigation) can now be managed independently, making the code easier to test, maintain, and extend.

## Appendix:

#### **Code Smells**

Code Smell	Description	Example	Suggested Solution
Long Methods	Methods that perform multiple tasks, making them hard to understand and maintain.	The updateInventory method combines inventory updating and file I/O logic.	Split updateInventory into smaller methods: one for inventory updating and another for file operations.
Nested Loops	Deeply nested loops reduce readability and increase complexity.	Nested looping over transactionItem and databaseItem.	Use a HashMap to cache databaseItem by itemID to minimize nested loops.

<b>Duplicate Code</b>	Repeated code	File reading and	Extract file handling logic
	for file handling	writing logic	into reusable helper
	and	appears in	methods.
	reading/writing	multiple places.	
	inventory data.		
Magic	Hardcoded	Hardcoded file	Define constants for file
Numbers/Strings	values or strings	paths like	paths and string keys
	used without	"inventory.txt"	(e.g.,
	explanation.	and string keys	FILE_PATH_INVENTORY,
		like "ItemID".	KEY_ITEM_ID).
Primitive	Overuse of	Using strings to	Use an enum for
Obsession	primitive types	represent itemID	transaction types and
	instead of	or transaction	consider using a class for
	meaningful	status instead of	inventory items.
	abstractions.	enums or	
		classes.	

#### **Performance Inefficiencies**

Issue	Description	Example	Suggested Solution
Repeated	Scanning the entire	Iterating through the	Use a
Object	databaseItem list	entire list for each	HashMap <integer,< th=""></integer,<>
Lookups	for every	transaction to find the	Item> to store
	transactionItem.	matching item.	databaseltem for
			constant-time
			lookups.
Redundant	Performing the	Repeated calls to	Store itemID in a local
Calculations	same calculation	getItemID() inside	variable before the
	repeatedly in a	nested loops.	loop to minimize
	loop.		method calls.
Inefficient	Using string	Using result += "Item:	Use StringBuilder for
String Ops	concatenation	" + itemName in a	concatenating strings
	inside loops.	loop.	inside loops to
			improve performance.
Excessive	Printing to the	Using	Replace
Logging	console	System.out.println for	System.out.println
	excessively instead		with a proper logging

of structured	error reporting in	framework like
logging.	catch blocks.	java.util.logging or
		Log4j.

## Housekeeping

Category	Issue	Fix
Code Smells	Duplicate file handling code.	Extract file reading/writing logic
		into dedicated helper methods.
	Combined logic in	Separate updateInventory into
	updateInventory that violates	modular methods for inventory
	single responsibility	processing and file updates.
	principle.	
Coding	Hardcoded strings like file	Define constants for all
Standards	paths and key names.	commonly used strings.
	Inconsistent error handling	Replace direct System.out.println
	(direct printing in catch	calls with proper logging using a
	blocks).	logging framework.
Performance	Repeated list traversal for	Cache databaseltem in a
Inefficiencies	inventory lookup.	HashMap for efficient lookups.
	Direct string concatenation	Use StringBuilder for string
	inside loops.	manipulations inside loops.

## **16. Java Code Review Checklist for payment\_interface**

File name	Payment_interface. java
class/interface	Payment_interface
name	

Category	Checklist	Yes/No	Issue	Fix
	Item			
Naming	Are class	Yes	None	None
Conventions	names			
	written in			
	PascalCase			
	?			
	Are	Yes	None	None
	variable			
	and method			
	names			
	written in			
	camelCase?			
	Are	No	No constants	Add constants for
	constants		used in the	button sizes, window
	written in		class.	size, etc.
	uppercase			
	with			
	underscore			
	s?			
Code Structure	Are access	Yes	None	None
	modifiers			
	used			
	correctly?			
	Are	Yes	None	None
	classes			
	and			
	interfaces			
	separated?			

Method Design	Are packages used appropriat ely?  Do methods	No No	Class is not part of any package.	Consider adding a package declaration.  Break
	have a single responsibi lity?		med() handles multiple actions.	actionPerformed() into smaller methods for clarity.
	Are method parameters limited?	Yes	None	None
	Is method overloadin g used properly?	Not Applica ble	No method overloading.	None
Exception Handling	Are exceptions handled with try-catch blocks?	No	No exception handling for invalid inputs.	Add exception handling, especially for user input during payments.
	Are specific exceptions used?	No	No specific exceptions are used.	Use specific exceptions (e.g., InvalidCardNumberExc eption).
Code Readability	Are comments added for complex logic?	No	Some areas of code lack detailed comments.	Add comments for clarity, especially in logic for payments and return handling.
	Is indentatio n consistent ?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None

	Are meaningful names used for variables, classes, and methods?	Yes	Names are meaningful.	None
Performance	Are data structures chosen based on performance?	No	No significant data structures involved.	None
	Are costly operations minimized in loops?	Not Applica ble	No loops present that require optimization.	None
	Is lazy initializa tion used?	Not Applica ble	No lazy initializati on used.	None
Memory Management	Are unnecessar y object references set to null?	No	No explicit memory management.	Consider nullifying unused references after use.
Security	Is user input validated?	No	Validation for cardNo and cash inputs is minimal.	Add validation for the payment inputs (e.g., card number format).
Maintainability	Are there long methods or deeply nested loops?	Yes	actionPerfor med() method is large.	Refactor actionPerformed() into smaller methods.
	Is there duplicated code?	Yes	Duplicate logic for handling cash and	Extract common logic into reusable methods.

		electronic payments.	
Are there any magic numbers?	Yes	Window size and button positions are hardcoded.	Replace magic numbers with constants.

Test Related Categories

Category	Checklist Item	Yes/No	Issu	Fi
			е	х
Test Coverage	Are unit tests provided for all public methods and critical functionalities?	No	N/A	N/ A
	Do unit tests cover edge cases and boundary values?	No	N/A	N/ A
Test Design	Are tests written following the AAA pattern?	Not Applicable	N/A	N/ A
	Are individual test cases independent?	Not Applicable	N/A	N/ A
	Are descriptive names used for test methods?	Not Applicable	N/A	N/ A
Assertions	Are assertions used to verify expected results?	Not Applicable	N/A	N/ A
	Are specific assertions used instead of general ones?	Not Applicable	N/A	N/ A
Boundary and Edge Cases	and Edge Are edge cases and boundary conditions tested?		N/A	N/ A
	Are invalid inputs covered by tests?	No	N/A	N/ A
Mocking and Stubbing	Are mocks or stubs used to isolate the unit under test?	No	N/A	N/ A
Performance Testing	Are tests to check performance for critical methods provided?	No	N/A	N/ A
Test Maintainability	Are test methods organized and modular?	Not Applicable	N/A	N/ A

	Is the	re a set	tup method	Not		N/A	N/
for initial		itializi	ng common	Appli	cable		A
	object	s?					
Housekeeping							
Category	Checklist	Yes/N	Issue	)		Fix	
	Item	О					
Code Smells	Are there	Yes	actionPerf	ormed	Refact	or	
	any code		() method	is	action	Perfor	med
	smells not		large and		() int	o smal	1er
	covered by		handles		method	s for	
	the		multiple t	asks.	readab	ility.	
	checklist?						
Coding	Are there	No	None		None		
Standards	any coding						
	standard						
	violations						
	not covered						
	by the						
	checklist?						
Performance	Are there	No	None		None		
Inefficienci	any						
es	performance						
	inefficienci						
	es not						
	covered by						
	the						
	checklist?						

#### Refactoring Task 1: Extract Repeated Code into a Helper Method

- Issue: The logic for disposing of the interface after completing an action (e.g., navigating to the Cashier\_Interface or Admin\_Interface) is repeated in multiple parts of the code.
- Refactoring Plan:
  - 1. Create a helper method called disposeInterface() that consolidates the repeated logic of setting visibility to false and disposing of the current interface.
  - 2. Replace all instances of the repeated code in the actionPerformed() method with calls to disposeInterface().
- **Benefit**: Reduces redundancy in the code, improving maintainability and readability. Any future changes to the disposal behavior can now be made in one place, reducing the risk of inconsistencies and making updates easier.

#### Refactoring Task 2: Consolidate UI Button Action Handling

- Issue: The actionPerformed() method contains repetitive blocks of code for handling actions related to different buttons (e.g., PayCash, PayElectronic, cancelTransaction, confirm). Each button action has similar logic for event handling, which could be consolidated.
- Refactoring Plan:
  - Extract the button-specific actions into smaller methods such as handleCashPayment(), handleElectronicPayment(), handleCancelTransaction(), and handleConfirmPayment().
  - 2. In the actionPerformed() method, instead of directly processing the logic for each button, call the appropriate handler method based on the source of the event.
- Benefit: Makes the code more modular and readable. Each button action is handled independently, making the code easier to test, extend, and maintain. This also improves scalability if new buttons or payment methods need to be added in the future.

#### Refactoring Task 3: Replace Magic Numbers with Named Constants

- **Issue**: The code uses hardcoded "magic numbers" for GUI layout and other values, such as button dimensions and screen positions, making it difficult to modify the layout or understand the purpose of these values.
- Refactoring Plan:
  - 1. Define meaningful constants at the top of the class for values like window size, button dimensions, and other commonly used numeric values (e.g., WINDOW WIDTH, BUTTON WIDTH, BUTTON HEIGHT).
  - 2. Replace the magic numbers in the layout code and other sections with these named constants.
- Benefit: Improves readability and maintainability by giving descriptive names to numeric values. If the layout or button sizes need to change, they can be modified in one place, reducing the likelihood of errors and making the code easier to update.

## Appendix:

#### Code Smells

Code Smell	Description	Example	Suggested Solution

Long Methods	Methods that perform multiple tasks, making them hard to understand and maintain.	The updateInventory method combines inventory updating and file I/O logic.	Split updateInventory into smaller methods: one for inventory updating and another for file operations.
Nested Loops	Deeply nested loops reduce readability and increase complexity.	Nested looping over transactionItem and databaseItem.	Use a HashMap to cache databaseItem by itemID to minimize nested loops.
Duplicate Code	Repeated code for file handling and reading/writing inventory data.	File reading and writing logic appears in multiple places.	Extract file handling logic into reusable helper methods.
Magic Numbers/Strings	Hardcoded values or strings used without explanation.	Hardcoded file paths like "inventory.txt" and string keys like "ItemID".	Define constants for file paths and string keys (e.g., FILE_PATH_INVENTORY, KEY_ITEM_ID).
Primitive Obsession	Overuse of primitive types instead of meaningful abstractions.	Using strings to represent itemID or transaction status instead of enums or classes.	Use an enum for transaction types and consider using a class for inventory items.

#### **Performance Inefficiencies**

Issue	Description	Example	Suggested Solution
Repeated	Scanning the entire	Iterating through the	Use a
Object	databaseItem list	entire list for each	HashMap <integer,< th=""></integer,<>
Lookups	for every	transaction to find the	Item> to store
	transactionItem.	matching item.	databaseItem for

			constant-time
			lookups.
Redundant	Performing the	Repeated calls to	Store itemID in a local
Calculations	same calculation	getItemID() inside	variable before the
	repeatedly in a	nested loops.	loop to minimize
	loop.		method calls.
Inefficient	Using string	Using result += "Item:	Use StringBuilder for
String Ops	concatenation	" + itemName in a	concatenating strings
	inside loops.	loop.	inside loops to
			improve performance.
Excessive	Printing to the	Using	Replace
Logging	console	System.out.println for	System.out.println
	excessively instead	error reporting in	with a proper logging
	of structured	catch blocks.	framework like
	logging.		java.util.logging or
			Log4j.

## Housekeeping

Category	Issue	Fix
Code Smells	Duplicate file handling code.	Extract file reading/writing logic
		into dedicated helper methods.
	Combined logic in	Separate updateInventory into
	updateInventory that violates	modular methods for inventory
	single responsibility	processing and file updates.
	principle.	
Coding	Hardcoded strings like file	Define constants for all
Standards	paths and key names.	commonly used strings.
	Inconsistent error handling	Replace direct System.out.println
	(direct printing in catch	calls with proper logging using a
	blocks).	logging framework.
Performance	Repeated list traversal for	Cache databaseltem in a
Inefficiencies	inventory lookup.	HashMap for efficient lookups.
	Direct string concatenation	Use StringBuilder for string
	inside loops.	manipulations inside loops.

## 17. Java Code Review Checklist for payment\_interface

File name	Transaction_interface.java
class/interface	Transaction_interface
name	

Category	Checklis	Yes/No	Issue	Fix
	t Item			
Naming Conventions	Are	Yes	None	None
	class			
	names			
	written			
	in			
	Pasca1Ca			
	se?			
	Are	Yes	None	None
	variable			
	and			
	method			
	names			
	written			
	in			
	camelCas			
	e?			
	Are	No	No constants	Add constants for
	constant		used in the	button sizes, window
	S		class.	size, etc.
	written			
	in			
	uppercas			
	e with			
	undersco			
	res?			
Code Structure	Are	Yes	None	None
	access			
	modifier			
	s used			
	correct1			
	у?			
	Are	Yes	None	None
	classes			
	and			

	interfac			
	es			
	separate d?			
	+	No	Class is not	Consider adding a
	Are	NO		Consider adding a
	packages		part of any	package declaration.
	used		package.	
	appropri			
	ately?			
Method Design	Do	No	actionPerfor	Refactor
	methods		med()	actionPerformed()
	have a		handles	into smaller methods
	single		multiple	for clarity.
	responsi		actions.	
	bility?			
	Are	Yes	None	None
	method			
	paramete			
	rs			
	limited?			
	Is	Not	No method	None
				None
	method	Applica	overloading.	
	overload	ble		
	ing used			
	properly			
	?			
Exception Handling	Are	No	No exception	Add exception
	exceptio		handling for	handling for user
	ns		invalid	inputs (e.g., phone
	handled		inputs.	number validation).
	with			
	try-			
	catch			
	blocks?			
	Are	No	No specific	Use specific
	specific		exceptions	exceptions where
	exceptio		are used.	applicable (e.g.,
	ns used?			InvalidPhoneNumberEx
	ns asca.			ception).
Code Readability	Are	No	Some areas	Add comments for
Code Readability	comments	110	lack	clarity, especially
	added		detailed	
				in logic for
	for		comments.	handling
				transactions.

	complex			
	logic?			
	Is	Yes	None	None
	indentat			
	ion			
	consiste			
	nt?			
	Are	Yes	None	None
	blank			
	lines			
	used to			
	separate			
	code			
	blocks?			
	Are	Yes	Names are	None
	meaningf		meaningful.	
	ul names		G	
	used for			
	variable			
	s,			
	classes,			
	and			
	methods?			
Performance	Are data	No	No	None
	structur		significant	
	es		data	
	chosen		structures	
	based on		involved.	
	performa			
	nce?			
	Are	Not	No loops	None
	costly	Applica	present that	
	operatio	ble	require	
	ns		optimization	
	minimize			
	d in			
	loops?			
	Is lazy	Not	No lazy	None
	initiali	Applica	initializati	
	zation	ble	on used.	
	used?			
Memory Management	Are	No	No explicit	Consider nullifying
	unnecess		memory	unused references
	ary		management.	after use.

	object				
	referenc				
	es set				
	to null?				
Security	Is user	No	Validation	Improve validation	
	input		for user	(e.g., for phone	
	validate		inputs is	number and coupon	
	d?		minimal.	code).	
Maintainability	Are	Yes	actionPerfor	Refactor	
	there		med() method	actionPerformed()	
	long		is large.	into smaller methods	
	methods			for better	
	or			maintainability.	
	deeply				
	nested				
	loops?				
	Is there	Yes	Duplicate	Extract common logic	
	duplicat		logic for	into reusable	
	ed code?		handling	methods.	
			different		
			operations		
			(e.g.,		
			handling		
			phone number		
			input).		
	Are	Yes	Window size	Replace magic	
	there		and button	numbers with	
	any		positions	constants.	
	magic		are		
	numbers?		hardcoded.		
Tost Polated Categories					

Test Related Categories

Category	Checklist Item	Yes/No	Issu	Fi
			е	х
Test Coverage	Are unit tests provided	No	N/A	N/
	for all public methods			A
	and critical			
	functionalities?			
	Do unit tests cover edge	No	N/A	N/
	cases and boundary			A
	values?			
Test Design	Are tests written	Not	N/A	N/
	following the AAA	Applicable		A
	pattern?			

	Are individual test	Not	N/A	N/
	cases independent?	Applicable		A
	Are descriptive names	Not	N/A	N/
	used for test methods?	Applicable		A
Assertions	Are assertions used to	Not	N/A	N/
	verify expected results?	Applicable		A
	Are specific assertions	Not	N/A	N/
	used instead of general	Applicable		A
	ones?			
Boundary and Edge	Are edge cases and	No	N/A	N/
Cases	boundary conditions			A
	tested?			
	Are invalid inputs	No	N/A	N/
	covered by tests?			A
Mocking and Stubbing	Are mocks or stubs used	No	N/A	N/
	to isolate the unit			A
	under test?			
Performance Testing	Are tests to check	No	N/A	N/
	performance for critical			A
	methods provided?			
Test Maintainability	Are test methods	Not	N/A	N/
	organized and modular?	Applicable		A
	Is there a setup method	Not	N/A	N/
	for initializing common	Applicable		A
	objects?			

Housekeeping

Category	Checklist	Yes/N	Issue	Fix
	Item	О		
Code Smells	Are there	Yes	actionPerformed	Refactor
	any code		() method is	actionPerformed
	smells not		large and	() into smaller
	covered by		handles	methods.
	the		multiple tasks.	
	checklist?			
Coding	Are there	No	None	None
Standards	any coding			
	standard			
	violations			
	not covered			
	by the			
	checklist?			
Performance	Are there	No	None	None
Inefficienci	any			
es	performance			

inefficienci	
es not	
covered by	
the	
checklist?	

# Refactoring Task 1: Simplify actionPerformed() Method by Extracting Logic

- Issue: The actionPerformed() method in Transaction\_Interface handles multiple actions and includes long if-else chains, making the code difficult to maintain.
- Refactoring Plan:
  - Break down the method by creating separate helper methods for each distinct action (e.g., handleAddItem(), handleRemoveItem(), handleEndTransaction(), handleCancelTransaction()).
  - 2. Move the relevant action logic into these methods.
  - 3. Modify actionPerformed() to delegate tasks to these helper methods.
- **Benefit:** Improves the readability and maintainability of the actionPerformed() method, making it easier to extend and modify. Each action will be easier to understand and modify independently.

#### Refactoring Task 2: Consolidate Customer Phone Number Validation

- **Issue**: The logic for validating the customer phone number in getCustomerPhone() is repeated and handled with a while loop that can be refactored for clarity.
- Refactoring Plan:
  - 1. Create a helper method, validatePhoneNumber(), which encapsulates the logic for input validation and re-prompting the user.
  - 2. Replace the validation code in getCustomerPhone() with calls to validatePhoneNumber().
- **Benefit**: Centralizes phone number validation logic in a single method, making the code cleaner and easier to update. If validation rules change in the future, they can be updated in one place.

# Refactoring Task 3: Replace Hardcoded Database File Paths with Constants

- **Issue**: The database file paths are hardcoded multiple times in the code, making the class difficult to maintain if the file paths change.
- Refactoring Plan:
  - 1. Define constants for the file paths at the top of the class, such as RENTAL DATABASE FILE, SALE DATABASE FILE, etc.
  - 2. Replace all instances of the hardcoded file paths with these constants.
- **Benefit**: Improves the maintainability of the code. File paths are now centralized, reducing the risk of errors and making it easier to change paths in the future.

## Appendix:

#### **Code Smells**

Code Smell	Description	Example	Suggested Solution
Long Methods	Methods that perform multiple tasks, making them hard to understand and maintain.	The updateInventory method combines inventory updating and file I/O logic.	Split updateInventory into smaller methods: one for inventory updating and another for file operations.
Nested Loops	Deeply nested loops reduce readability and increase complexity.	Nested looping over transactionItem and databaseItem.	Use a HashMap to cache databaseItem by itemID to minimize nested loops.
Duplicate Code	Repeated code for file handling and reading/writing inventory data.	File reading and writing logic appears in multiple places.	Extract file handling logic into reusable helper methods.

Magic	Hardcoded	Hardcoded file Define constants for f	
Numbers/Strings	values or strings	paths like	paths and string keys
	used without	"inventory.txt"	(e.g.,
	explanation.	and string keys	FILE_PATH_INVENTORY,
		like "ItemID".	KEY_ITEM_ID).
Primitive	Overuse of	Using strings to	Use an enum for
Obsession	primitive types	represent itemID	transaction types and
	instead of	or transaction	consider using a class for
	meaningful	status instead of	inventory items.
	abstractions.	enums or	
		classes.	

## **Performance Inefficiencies**

Issue	Description	Example	Suggested Solution
Repeated	Scanning the entire	Iterating through the	Use a
Object	databaseItem list	entire list for each	HashMap <integer,< th=""></integer,<>
Lookups	for every	transaction to find the	Item> to store
	transactionItem.	matching item.	databaseltem for
			constant-time
			lookups.
Redundant	Performing the	Repeated calls to	Store itemID in a local
Calculations	same calculation	getItemID() inside	variable before the
	repeatedly in a	nested loops.	loop to minimize
	loop.		method calls.
Inefficient	Using string	Using result += "Item:	Use StringBuilder for
String Ops	concatenation	" + itemName in a	concatenating strings
	inside loops.	loop.	inside loops to
			improve performance.
Excessive	Printing to the	Using	Replace
Logging	console	System.out.println for	System.out.println
	excessively instead	error reporting in	with a proper logging
	of structured	catch blocks.	framework like
	logging.		java.util.logging or
			Log4j.

## Housekeeping

Category	Issue	Fix
Code Smells	Duplicate file handling code.	Extract file reading/writing logic
		into dedicated helper methods.
	Combined logic in	Separate updateInventory into
	updateInventory that violates	modular methods for inventory
	single responsibility	processing and file updates.
	principle.	
Coding	Hardcoded strings like file	Define constants for all
Standards	paths and key names.	commonly used strings.
	Inconsistent error handling	Replace direct System.out.println
	(direct printing in catch	calls with proper logging using a
	blocks).	logging framework.
Performance	Repeated list traversal for	Cache databaseltem in a
Inefficiencies	inventory lookup.	HashMap for efficient lookups.
	Direct string concatenation	Use StringBuilder for string
	inside loops.	manipulations inside loops.

# 18. Java Code Review Checklist for Update\_Employee interface

File name	Update_Employee interface.java
class/interface	Update_Employee interface
name	

Category	Checklist	Yes/No	Issue	Fix
	Item			
Naming	Are class	Yes	None	None
Conventions	names			
	written in			
	PascalCase?			
	Are	Yes	None	None
	variable			
	and method			
	names			

	written in			
	camelCase? Are constants written in uppercase with underscores ?	No	No constants used in the class.	Add constants for button sizes, window size, etc.
Code Structure	Are access modifiers used correctly?	Yes	None	None
	Are classes and interfaces separated?	Yes	None	None
	Are packages used appropriate ly?	No	Class is not part of any package.	Consider adding a package declaration.
Method Design	Do methods have a single responsibil ity?	Yes	None	None
	Are method parameters limited?	Yes	None	None
	Is method overloading used properly?	Not Applica ble	No method overloadin g.	None
Exception Handling	Are exceptions handled with try- catch blocks?	No	No exception handling for invalid inputs.	Add exception handling for user inputs (e.g., username, password).
	Are specific exceptions used?	No	No specific exceptions are used.	Use specific exceptions where applicable (e.g.,

				InvalidUsernameExcep tion).
Code Readability	Are comments added for complex logic?	No	Some areas lack detailed comments.	Add comments for clarity, especially in logic for updating employee information.
	Is indentation consistent?	Yes	None	None
	Are blank lines used to separate code blocks?	Yes	None	None
	Are meaningful names used for variables, classes, and methods?	Yes	Names are meaningful .	None
Performance	Are data structures chosen based on performance ?	No	No significan t data structures involved.	None
	Are costly operations minimized in loops?	Not Applica ble	No loops present that require optimizati on.	None
	Is lazy initializat ion used?	Not Applica ble	No lazy initializa tion used.	None
Memory Management	Are unnecessary object references set to null?	No	No explicit memory management	Consider nullifying unused references after use.

Security	Is user input validated?	No	Validation for user inputs is minimal.	Improve validation (e.g., for username, password).
Maintainability	Are there long methods or deeply nested loops?	No	None	None
	Is there duplicated code?	Yes	Duplicate logic for handling employee update.	Extract common logic into reusable methods.
	Are there any magic numbers?	Yes	Window size and button positions are hardcoded.	Replace magic numbers with constants.

Test Related Categories

Category Checklist Item		Yes/No	Issu	Fi
			е	X
Test Coverage	Are unit tests provided	No	N/A	N/
	for all public methods			A
	and critical			
	functionalities?			
	Do unit tests cover edge	No	N/A	N/
	cases and boundary			A
	values?			
Test Design	Are tests written	Not	N/A	N/
	following the AAA	Applicable		A
	pattern?			
	Are individual test	Not	N/A	N/
	cases independent?	Applicable		A
	Are descriptive names	Not	N/A	N/
	used for test methods?	Applicable		A
Assertions	Are assertions used to	Not	N/A	N/
	verify expected results?	Applicable		A
	Are specific assertions	Not	N/A	N/
	used instead of general	Applicable		A
	ones?			

Boundary and Edge	Are edge cases and	No	N/A	N/
Cases	boundary conditions			A
	tested?			
	Are invalid inputs	No	N/A	N/
	covered by tests?			A
Mocking and Stubbing	Are mocks or stubs used	No	N/A	N/
	to isolate the unit			A
	under test?			
Performance Testing	mance Testing   Are tests to check		N/A	N/
	performance for critical			A
	methods provided?			
Test Maintainability	Are test methods	Not	N/A	N/
	organized and modular?	Applicable		A
	Is there a setup method	Not	N/A	N/
	for initializing common	Applicable		A
	objects?			
	TT 1 •			

Housekeeping

Category	Checklist	Yes/N	Issue	Fix
	Item	0		
Code Smells	Are there	Yes	actionPerformed	Refactor
	any code		() method is	actionPerformed
	smells not		large and	() into smaller
	covered by		handles	methods.
	the		multiple tasks.	
	checklist?			
Coding	Are there	No	None	None
Standards	any coding			
	standard			
	violations			
	not covered			
	by the			
	checklist?			
Performance	Are there	No	None	None
Inefficienci	any			
es	performance			
	inefficienci			
	es not			
	covered by			
	the			
	checklist?			

# Refactoring Task 1: Extract Repeated Code for Interface Disposal into a Helper Method

 Issue: The logic for disposing of the interface after an action is completed is repeated in multiple places (for example, after updating employee details or exiting).

#### • Refactoring Plan:

- 1. Create a helper method called disposeInterface(). This method should handle the logic of setting visibility to false and disposing of the interface.
- 2. Replace the repeated code in the actionPerformed() method with calls to disposeInterface().
- 3. Update actionPerformed() to call disposeInterface(this) when needed.
- **Benefit**: Reduces redundant code, making the class easier to maintain. If the disposal logic needs to change, it can be done in one location rather than multiple places.

## Refactoring Task 2: Replace Magic Numbers with Constants

- **Issue**: There are hardcoded values for window size, button positions, and component dimensions (e.g., button width, height, and position) scattered throughout the code.
- Refactoring Plan:
  - 1. Define constants at the top of the class to represent these magic numbers (e.g., window\_width, window\_Height, Button\_width, Button Height).
  - 2. Replace all instances of these hardcoded values in the layout setup with the defined constants.
  - 3. Replace all layout and button position logic with these constants.
- **Benefit**: Improves code readability and maintainability by giving meaningful names to numeric values. Changes to layout configuration can now be done in one place.

# Refactoring Task 3: Break Down Large actionPerformed() Method into Smaller Methods

- **Issue:** The actionPerformed() method is lengthy and handles multiple responsibilities, such as handling button clicks, updating employee information, displaying error messages, and navigating to other interfaces.
- Refactoring Plan:

- 1. Extract the logic for each action into separate, well-named methods, such as:
  - handleEnterButtonClick()
  - handleExitButtonClick()
  - navigateToAdminInterface()
  - showErrorMessage(String message)
- 2. In the actionPerformed() method, delegate each action to its corresponding method based on the event source.
- **Benefit**: Enhances code readability, makes it easier to follow and debug, and supports better unit testing. Each method now has a clear, single responsibility, making the code easier to maintain and extend.

# Appendix:

#### **Code Smells**

Code Smell	Description	Example	Suggested Solution
Long Methods	Methods that perform multiple tasks, making them hard to understand and maintain.	The updateInventory method combines inventory updating and file I/O logic.	Split updateInventory into smaller methods: one for inventory updating and another for file operations.
Nested Loops	Deeply nested loops reduce readability and increase complexity.	Nested looping over transactionItem and databaseItem.	Use a HashMap to cache databaseItem by itemID to minimize nested loops.
Duplicate Code	Repeated code for file handling and reading/writing inventory data.	File reading and writing logic appears in multiple places.	Extract file handling logic into reusable helper methods.
Magic Numbers/Strings	Hardcoded values or strings	Hardcoded file paths like "inventory.txt"	Define constants for file paths and string keys (e.g.,

	used without	and string keys	FILE_PATH_INVENTORY,
	explanation.	like "ItemID".	KEY_ITEM_ID).
Primitive	Overuse of	Using strings to	Use an enum for
Obsession	primitive types	represent itemID	transaction types and
	instead of	or transaction	consider using a class for
	meaningful	status instead of	inventory items.
	abstractions.	enums or	
		classes.	

### **Performance Inefficiencies**

Issue	Description	Example	Suggested Solution
Repeated	Scanning the entire	Iterating through the	Use a
Object	databaseItem list	entire list for each	HashMap <integer,< th=""></integer,<>
Lookups	for every	transaction to find the	Item> to store
	transactionItem.	matching item.	databaseItem for
			constant-time
			lookups.
Redundant	Performing the	Repeated calls to	Store itemID in a local
Calculations	same calculation	getItemID() inside	variable before the
	repeatedly in a	nested loops.	loop to minimize
	loop.		method calls.
Inefficient	Using string	Using result += "Item:	Use StringBuilder for
String Ops	concatenation	" + itemName in a	concatenating strings
	inside loops.	loop.	inside loops to
			improve performance.
Excessive	Printing to the	Using	Replace
Logging	console	System.out.println for	System.out.println
	excessively instead	error reporting in	with a proper logging
	of structured	catch blocks.	framework like
	logging.		java.util.logging or
			Log4j.

# Housekeeping

Category	Issue	Fix

Code Smells	Duplicate file handling code.	Extract file reading/writing logic into dedicated helper methods.
		into dedicated netper methods.
	Combined logic in	Separate updateInventory into
	updateInventory that violates	modular methods for inventory
	single responsibility	processing and file updates.
	principle.	
Coding	Hardcoded strings like file	Define constants for all
Standards	paths and key names.	commonly used strings.
	Inconsistent error handling	Replace direct System.out.println
	(direct printing in catch	calls with proper logging using a
	blocks).	logging framework.
Performance	Repeated list traversal for	Cache databaseltem in a
Inefficiencies	inventory lookup.	HashMap for efficient lookups.
	Direct string concatenation	Use StringBuilder for string
	inside loops.	manipulations inside loops.

# 19. Java Code Review Checklist for Employee

File name	Employee. java
class/interface	Employee
name	

Category	Checklist Item	Yes/No	Issue	Fix
Naming	Are class names	Yes	None	None
Conventions	written in			
	PascalCase?			
	Are variable and	Yes	None	None
	method names			
	written in			
	camelCase?			
	Are constants	No	No constants	Add
	written in		used in the	constants if
	uppercase with		class.	required.
	underscores?			
Code Structure	Are access	Yes	None	None
	modifiers used			
	correctly?			

	Are methods well-	Yes	None	None		
	structured and not					
	too long?					
Method Design	Do methods have a	Yes	None	None		
	single					
	responsibility?	77	NT.	NT		
	Are method	Yes	None	None		
	parameters limited					
	to necessary					
	inputs? Is method	No	No	None		
	overloading used	NO	overloading	None		
	properly?		used.			
Code	Are comments added	No	No comments	Add comments		
Readability	for complex logic?	110	for methods.	for methods		
1100000 1110				for clarity.		
	Is indentation	Yes	None	None		
	consistent?					
	Are meaningful	Yes	Names are	None		
	names used for		meaningful.			
	variables,					
	classes, and					
	methods?					
Security	Are user inputs	No	No input	Add		
	validated?		validation	validation		
			(e.g.,	checks to		
			username,	ensure data		
			password).	integrity.		
Maintainability	Are there long	No	None	None		
	methods or deeply					
	nested loops?	NT.	NT .	NT .		
	Is there duplicated code?	No	None	None		
	Are there any	No	No magic	None		
	magic numbers?	110	numbers in	None		
	magic namedis.		this class.			
	Test Related Categories					
Category	Checklist Item	Yes/No	Issue	Fix		
Test Coverage	Are unit tests	Yes	Full coverage	Add tests		
	provided for all		of getter and	for edge		
	public methods and		setter	cases.		
		1	1	i II		
	critical		methods.			

	cover	it tests edge cases oundary s?	No	tested	cases no d (e.g., strings, values).	for case	n as
Test Design		ests written wing the AAA rn?	Yes	AAA (A	en using Arrange, Assert)	None	
	test	ndividual cases endent?	Yes	Tests indep	are endent.	None	)
	names	escriptive used for methods?	Yes	names	method are iptive.	None	9
Assertions	used	ssertions to verify ted results?	Yes		tions ard in all cases.	e None	9
Boundary and Edge Cases	bound	dge cases and ary tions tested?	No	tested very	tions node (e.g., long empty	t for bour	tests ndary edge es.
		nvalid inputs ed by tests?	Yes	tests wrong	id input (e.g., ame) are ed.	None	)
Mocking and Stubbing	used	ocks or stubs to isolate nit under	No	Not applicable, as there is no external system interaction.			
Test Maintainability			Yes	Test methods None are organized.		)	
		Houseke			37 /37	-	D.
			Issue None	None			

Coding Standards	Are there any coding standard	No	None	None	
	violations not covered by the				
	checklist?				
Performance	Are there any performance	No	None	None	
Inefficiencies	inefficiencies not covered by				
	the checklist?				

## Refactoring Task 1: Add Input Validation for Employee Attributes

- **Issue**: The Employee class lacks input validation for the attributes, particularly the username, password, and position fields.
- Refactoring Plan:
  - 1. Implement input validation for the username, password, and position fields to ensure that they meet specific criteria (e.g., non-empty, valid length, correct format).
  - 2. Update the constructor and setter methods to include validation checks before assigning the values to the fields.
  - 3. Throw custom exceptions (e.g., InvalidUsernameException, InvalidPasswordException) if the input doesn't meet the required conditions.
- **Benefit**: Adds data integrity and security to the application by preventing invalid or malicious input. It improves the robustness of the class and ensures that only valid data is accepted.

## Refactoring Task 2: Consolidate Setter Methods

- **Issue**: The Employee class has separate setter methods for name, position, and password. While this is not inherently wrong, it could be simplified to make the code cleaner and more flexible.
- Refactoring Plan:
  - Create a single setter method, updateEmployeeInfo(String name, String position, String password), which updates the name, position, and password fields all at once.
  - 2. Replace the individual setter calls with this new consolidated method in places where multiple attributes need to be updated at once.
- Benefit: Simplifies the code and reduces the number of method calls, especially in scenarios where multiple employee attributes need to be updated simultaneously. It also improves maintainability, making it easier to update or expand in the future.

## Refactoring Task 3: Extract Password Handling Logic

- **Issue:** The handling of the password attribute is done directly in the Employee class without any abstraction. This may cause issues if more complex password-related logic (e.g., encryption, validation) is introduced later.
- Refactoring Plan:
  - 1. Extract the password handling logic into a separate class, e.g.,

    PasswordManager, which handles password validation and hashing.
  - 2. Modify the Employee class to delegate password-related logic to the PasswordManager class, such as validating or setting the password.
  - 3. Implement the PasswordManager to include methods like isValidPassword(String password) and hashPassword(String password).
- **Benefit**: Improves maintainability and scalability by isolating password-related logic in a dedicated class. It also prepares the code for future enhancements (e.g., adding password encryption) without bloating the Employee class with concerns outside of its core responsibility.

# Appendix:

#### **Code Smells**

Code Smell	Description	Example	Suggested Solution
Long Methods	Methods that	The	Split updateInventory into
	perform	updateInventory	smaller methods: one for
	multiple tasks,	method	inventory updating and
	making them	combines	another for file
	hard to	inventory	operations.
	understand and	updating and file	
	maintain.	I/O logic.	
Nested Loops	Deeply nested	Nested looping	Use a HashMap to cache
	loops reduce	over	databaseItem by itemID
	readability and	transactionItem	to minimize nested loops.
	increase	and	
	complexity.	databaseltem.	
<b>Duplicate Code</b>	Repeated code	File reading and	Extract file handling logic
	for file handling	writing logic	into reusable helper
	and		methods.

	reading/writing	appears in	
	inventory data.	multiple places.	
Magic	Hardcoded	Hardcoded file	Define constants for file
Numbers/Strings	values or strings	paths like	paths and string keys
	used without	"inventory.txt"	(e.g.,
	explanation.	and string keys	FILE_PATH_INVENTORY,
		like "ItemID".	KEY_ITEM_ID).
Primitive	Overuse of	Using strings to	Use an enum for
Obsession	primitive types	represent itemID	transaction types and
	instead of	or transaction	consider using a class for
	meaningful	status instead of	inventory items.
	abstractions.	enums or	
		classes.	

## **Performance Inefficiencies**

Issue	Description	Example	Suggested Solution
Repeated	Scanning the entire	Iterating through the	Use a
Object	databaseItem list	entire list for each	HashMap <integer,< th=""></integer,<>
Lookups	for every	transaction to find the	Item> to store
	transactionItem.	matching item.	databaseItem for
			constant-time
			lookups.
Redundant	Performing the	Repeated calls to	Store itemID in a local
Calculations	same calculation	getItemID() inside	variable before the
	repeatedly in a	nested loops.	loop to minimize
	loop.		method calls.
Inefficient	Using string	Using result += "Item:	Use StringBuilder for
String Ops	concatenation	" + itemName in a	concatenating strings
	inside loops.	loop.	inside loops to
			improve performance.
Excessive	Printing to the	Using	Replace
Logging	console	System.out.println for	System.out.println
	excessively instead	error reporting in	with a proper logging
	of structured	catch blocks.	framework like
	logging.		java.util.logging or
			Log4j.
	<u> </u>		

# Housekeeping

Category	Issue	Fix	
Code Smells Duplicate file handling code.		Extract file reading/writing logic	
		into dedicated helper methods.	
	Combined logic in	Separate updateInventory into	
	updateInventory that violates	modular methods for inventory	
	single responsibility	processing and file updates.	
	principle.		
Coding	Hardcoded strings like file	Define constants for all	
Standards	paths and key names.	commonly used strings.	
	Inconsistent error handling	Replace direct System.out.println	
	(direct printing in catch	calls with proper logging using a	
	blocks).	logging framework.	
Performance	Repeated list traversal for	Cache databaseltem in a	
Inefficiencies	inventory lookup.	HashMap for efficient lookups.	
	Direct string concatenation	Use StringBuilder for string	
	inside loops.	manipulations inside loops.	