Module 1: The AI-Native Smells Detector, Lesson 2: Fragility | Assignment: The Quiet Coupling

Overview

A small Order class is used in three places. None of those places look risky. But the class has a hidden coupling that will break all three the moment someone touches an unrelated internal detail. Read it, find it, and propose a fix.

Estimated Time

10 minutes

Before You Start

- Read the code block below.
- Have a blank text file or notepad ready for your answers.
- You may run the code mentally; you do not need to execute it.

REMEMBER

Fragility is the smell of unwritten contracts. Look for code that reaches across module boundaries without going through a method or an explicit type.

Your Task

Read this code carefully. Identify the LINE that is the source of the fragility, then answer the questions in the Submit section.

```
# order.py
class Order:
    def __init__(self, order_id):
        self.order_id = order_id
        self._items = []   # leading underscore is convention, not enforcement
        self._status = "open"

    def add_item(self, name, price):
        self._items.append({"name": name, "price": price})


# reports.py - reads from Order
def order_total(order):
    return sum(item["price"] for item in order._items)
```

```
# emails.py - reads from Order
def receipt_lines(order):
    return [f"{i['name']}: ${i['price']:.2f}" for i in order._items]


# analytics.py - reads from Order
def item_count(order):
    return len(order._items)
```

What to Submit

1. Name the smell and the SPECIFIC mechanism (one sentence each).
2. If a future engineer changes the Order class to store items as a list of tuples (name, price) instead of dicts, which modules break? Why?
3. Propose ONE small change to the Order class that would prevent ALL three downstream modules from being affected by the storage format. Describe it in one or two sentences; you do not need to write the code.
4. Bonus: write the SIGNATURE (not the body) of the new method or property you would add to Order.

Solution Sketch (Instructor)

SOLUTION

Smell: fragility. Mechanism: three modules reach into Order._items directly, depending on its storage format (a list of dicts). Modules that break: all three - reports.py via item['price'], emails.py via i['name'] and i['price'], analytics.py via len(). Fix: add a public method like 'items()' that returns an immutable, well-typed view (e.g., a tuple of LineItem dataclasses) and update callers to use it. Future internal storage changes are then invisible to callers. Signature: 'def items(self) -> tuple[LineItem, ...]'.

Source File Reference

This assignment is part of Lesson 2 of Module 1: The AI-Native Smells Detector. The code snippet above is a teaching example. The full hands-on for this lesson uses related source files in the source_code/ directory.