Module 1: The AI-Native Smells Detector, Lesson 5: Needless Complexity | Assignment: The Three-Question Audit

Overview

Below is a class hierarchy that an AI assistant generated when asked for 'a robust, production-grade way to send a notification.' Apply the three diagnostic questions from the lecture and decide which parts deserve to exist.

Estimated Time

10 minutes

Before You Start

- Read the code block below.
- Have a blank text file or notepad ready for your answers.
- The behavior the user actually asked for: 'send a notification.' That is it. Keep that fact in front of you.

REMEMBER

The diagnostic: (1) What does this abstraction protect me from? (2) When has that thing actually happened? (3) How many concrete subtypes use this today? Vague + never + one = needless.

Your Task

Read the AI-generated hierarchy. Apply the three diagnostic questions. Answer the prompts in the Submit section.

```
# notifier.py - AI-generated 'production-grade notification system'

from abc import ABC, abstractmethod
from dataclasses import dataclass

@dataclass(frozen=True)
class NotificationPayload:
    body: str

class INotificationStrategy(ABC):
    @abstractmethod
    def send(self, payload: NotificationPayload) -> "NotificationResult": ...
```

```
class ConsoleNotificationStrategy(INotificationStrategy):
    def send(self, payload):
        print(payload.body)
        return NotificationResult(success=True)

@dataclass(frozen=True)
class NotificationResult:
    success: bool

class NotificationStrategyFactory:
    @staticmethod
    def create(kind: str) -> INotificationStrategy:
        if kind == "console":
            return ConsoleNotificationStrategy()
        raise ValueError(kind)

@dataclass
class NotificationConfig:
    strategy_kind: str = "console"

class NotificationService:
    def __init__(self, config: NotificationConfig):
        self._strategy = NotificationStrategyFactory.create(config.strategy_kind)
    def notify(self, message: str) -> NotificationResult:
        return self._strategy.send(NotificationPayload(body=message))

# Usage:
# svc = NotificationService(NotificationConfig())
# svc.notify("hello")
```

What to Submit

1. Apply the three diagnostic questions to ONE of the classes (your choice). Write each answer in one short sentence.
2. Identify every abstraction in the file that has exactly ONE concrete subtype today.
3. Rewrite the entire 'send a notification' behavior in three lines or fewer. (You may use the standard library.)
4. Bonus: write the prompt you would give the AI assistant to get the three-line version on the first try.

Solution Sketch (Instructor)

SOLUTION

Diagnostic example for INotificationStrategy: (1) it would protect against having to swap notification mechanisms; (2) it never has - there is only one mechanism; (3) one concrete

subtype. Vague, never, one - needless. Single-subtype abstractions: INotificationStrategy, NotificationStrategyFactory, NotificationConfig, NotificationService - all have one concrete use. Three-line rewrite: 'def notify(message: str) -> None: print(message)'. Bonus prompt: 'Write the simplest possible Python function to print a notification message. No classes, no patterns, no config objects. Just a function.'

Source File Reference

This assignment is part of Lesson 5 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.