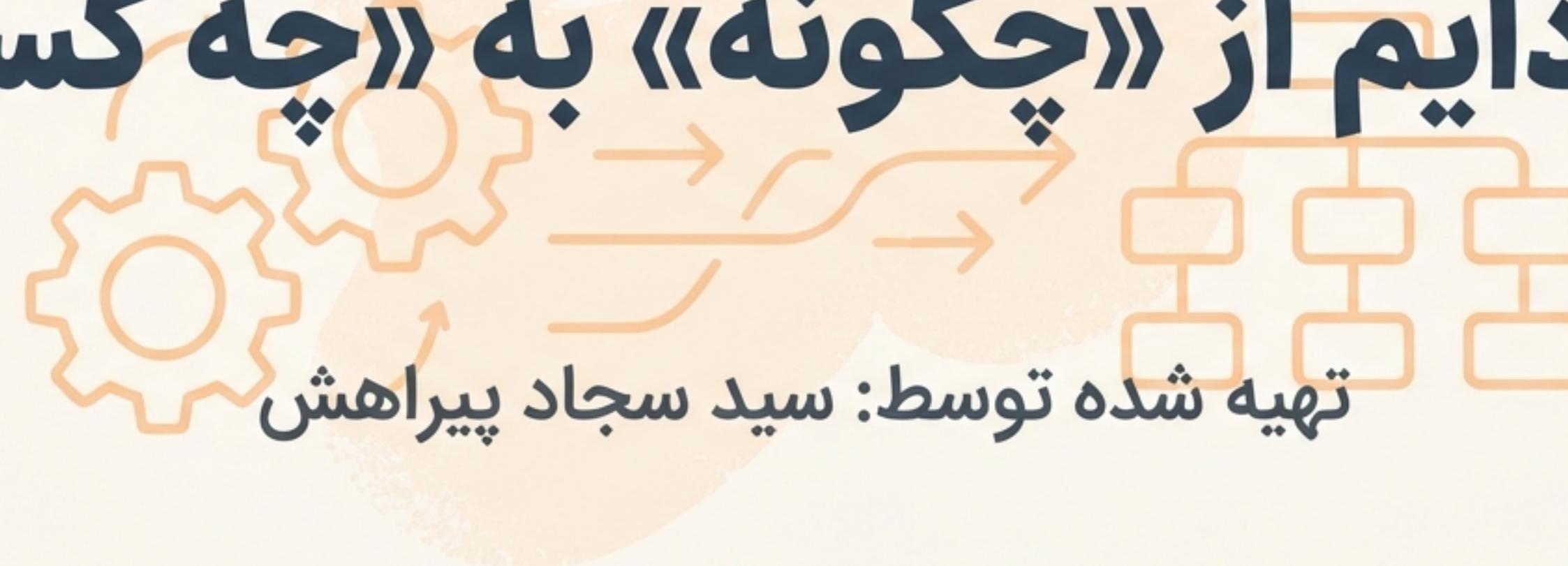


بخش ۷: تفکر شیء‌گرا – تغییر پارادایم از «چگونه» به «چه کسی»



تهریه شده توسط: سید سجاد پیراهاش

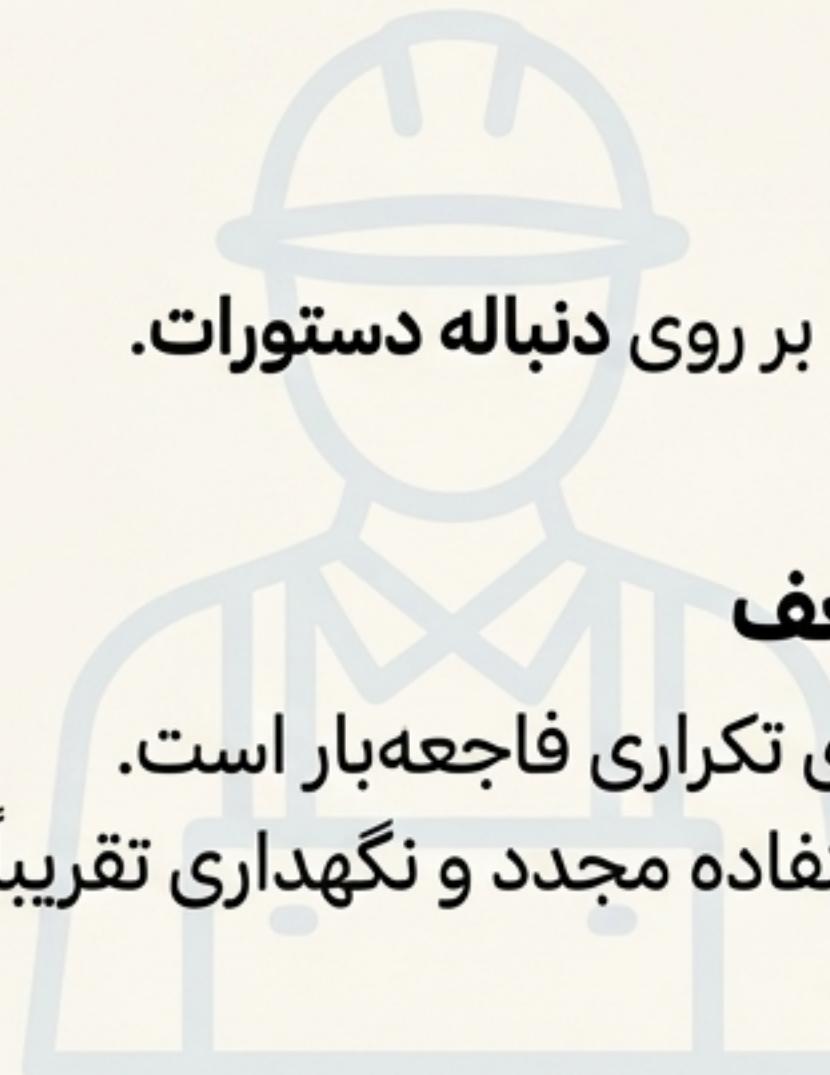
سفر یک برنامه‌نویس: از کارگر تنها تا مدیر یک تیم متخصص

قبل از یادگیری قواعد شیء‌گرایی، باید طرز فکر خود را تغییر دهیم. این یک سفر برای تکامل ذهنیت ماست. ما از یک مجری ساده دستورات، به یک صنعتگر ماهر، و در نهایت به یک مدیر تبدیل می‌شویم که تیمی از متخصصان (اشیاء) را رهبری می‌کند.



مرحله اول: ذهنیت خطی – « فقط انجامش بده »

برنامه یک دستور پخت طولانی و یکپارچه است. کامپیوتر از خط اول شروع می‌کند و بدون انحراف تا آخر می‌رود.



فلسفه

تمرکز کامل بر روی دنباله دستورات.

نقطه ضعف

برای کارهای تکراری فاجعه‌بار است.
قابلیت استفاده مجدد و نگهداری تقریباً صفر است.

‘Sum2NumLinear.java’

```
// 1. Define first number  
int number1 = 10;  
// 2. Define second number  
int number2 = 20;  
// 3. Perform addition  
int result = number1 + number2;  
// 4. Print the result  
System.out.println("The sum is: " + result);
```

مرحله دوم: ذهنیت رویه‌ای – «جعبه ابزار خودت را بساز»

وظایف بزرگ را به زیروظیفه‌های کوچک‌تر و قابل استفاده مجدد به نام تابع (Function) تقسیم کن.

‘Sum2NumProcedural.java’

فلسفه

تمرکز بر روی «افعال» و «عملیات». یک پیشرفت بزرگ در استفاده مجدد از کد.

نقطه ضعف کلیدی

داده‌ها از عملیات جدا هستند. این جدایی در سیستم‌های بزرگ، بزرگ، منشأ اصلی خطا و خطا و پیچیدگی است.

```
// Reusable tool (Function)
public static int add(int num1, int num2) { ← ابزار قابل استفاده مجدد
    return num1 + num2;
}

public static void main(String[] args) {
    int number1 = 10;
    int number2 = 20;
    // Using the tool
    int result = add(number1, number2);
    System.out.println("The sum is: " + result);
}
```

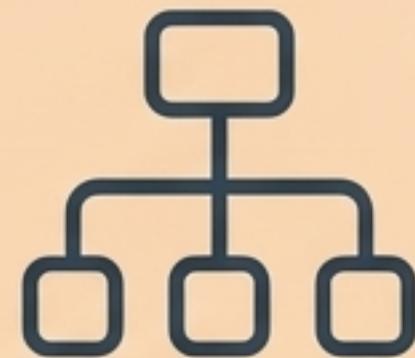
نقطه شکست: وقتی جعبه ابزار به «کد اسپاگتی» تبدیل می‌شود



در پروژه‌های بزرگ، جدایی داده از رفتار فاجعه می‌آفریند. توابع مختلف شروع به دستکاری داده‌های مشترک (Global State) نتیجه؟ سیستمی شکننده، غیرقابل پیش‌بینی و کابوسی برای نگهداری. هر تغییر کوچکی می‌تواند کل سیستم را خراب کند.

مرحله سوم: ذهنیت شیءگرا – «یک تیم متخصص استخدام کن»

دنیا را به صورت مجموعه‌ای از اشیاء (Objects) هوشمند و مستقل مدل‌سازی کن.
هر شیء هم «دانش» (داده‌ها) و هم «توانایی» (رفتارها)ی خود را دارد.



سوال رویه‌ای:

چگونه این کار را انجام
دهم؟

سوال شیءگرا:

چه کسی مسئول انجام این
کار است؟

`Calculator.java`

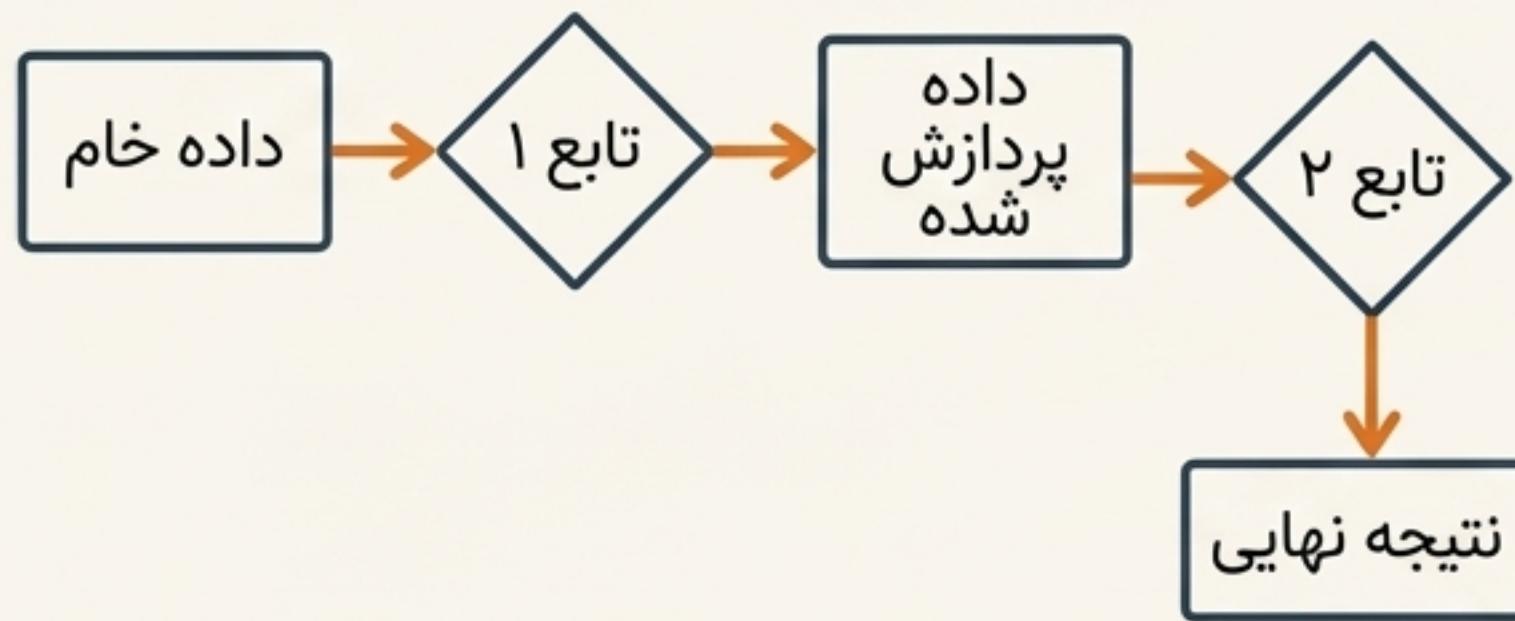
```
// The Specialist  
public class Calculator { ←  
    // Behavior owned by the specialist  
    public int add(int num1, int num2) {  
        return num1 + num2;  
    }  
}  
  
// Manager delegates the task  
Calculator myCalc = new Calculator();  
int result = myCalc.add(10, 20);  
System.out.println("The sum is: " + result);
```

متخصص مسئول

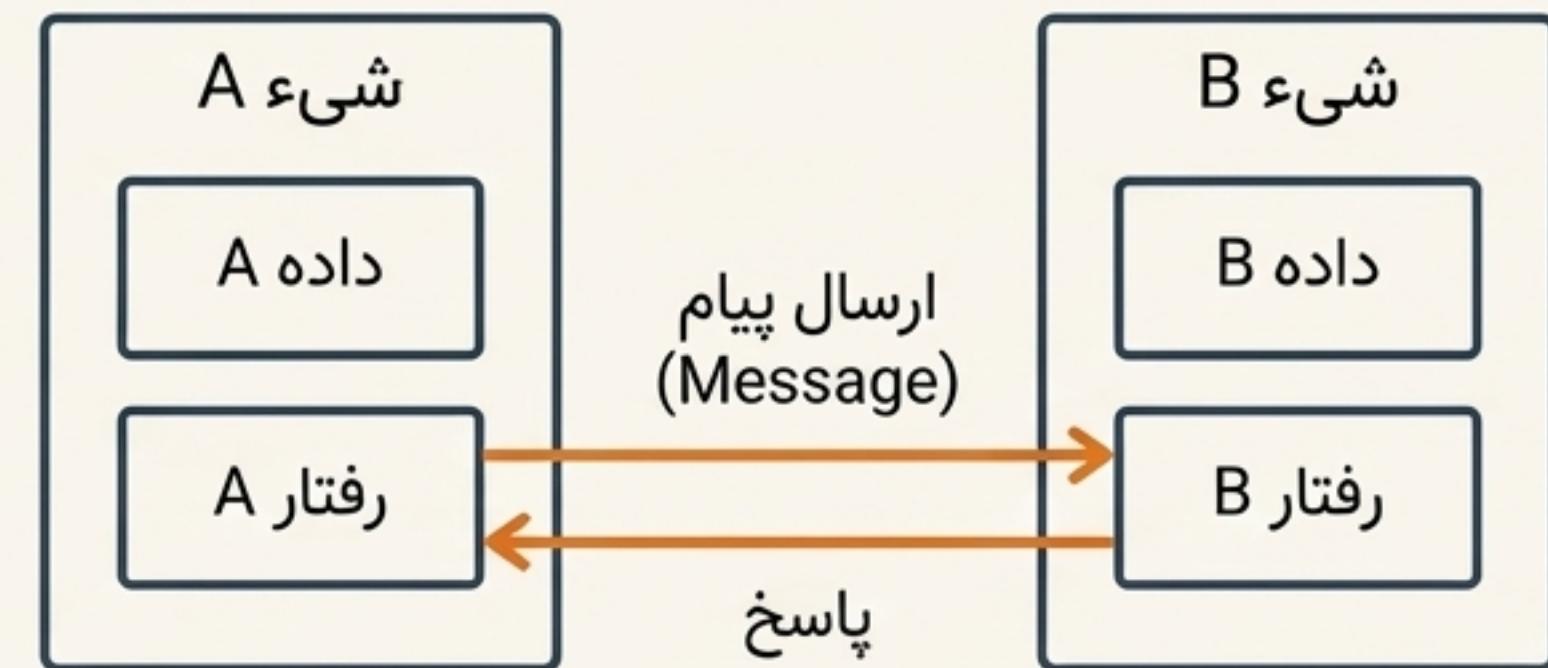
نقشه ذهنی دو پارادایم: تفاوت در جریان کنترل

تفاوت اصلی در نحوه جریان یافتن منطق در برنامه است. در تفکر رویه‌ای، ما داده‌ها را از یک تابع به تابع دیگر پاس می‌دهیم. در تفکر شی‌عگرا، اشیاء با ارسال پیام، از یکدیگر درخواست انجام کار می‌کنند.

جریان رویه‌ای (داده‌ها منفعل هستند)



جریان شی‌عگرا (اشیاء فعال هستند)



تکامل یک وظیفه ساده: سه نگاه به «جمع دو عدد»

ذهنیت « فقط انجامش بده »

```
int number1 = 10;  
int number2 = 20;  
int result = number1 + number2;  
System.out.println(result);
```

یک دستور پخت یکبار مصرف.

ذهنیت « یک ابزار بساز »

```
public static int add(int a, int b) {  
    return a + b;  
}  
  
int result = add(10, 20);
```

ابزار جدا از داده‌ها.

ذهنیت « یک متخصص استخدام کن »

```
public class Calculator {  
    public int add(int a, int b){...}  
}  
  
Calculator c = new Calculator();  
int result = c.add(10, 20);
```

متخصص مسئول رفتار.

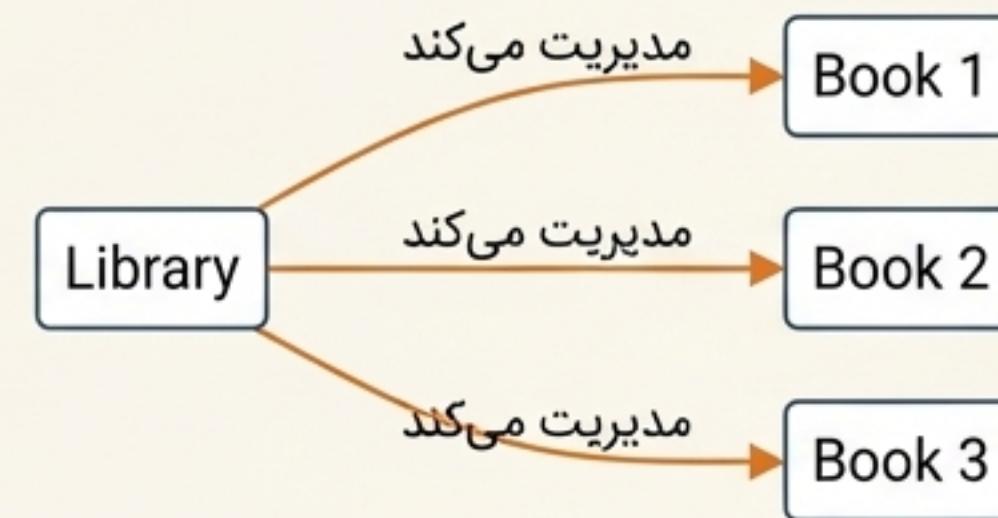
مقایسه جامع پارادایم‌ها: یک نگاه کلی

معیار	Linear / Monolithic	Procedural / Modular	Object-Oriented (OOP)
واحد اصلی	کل برنامه	تابع / رویه	شیء / کلاس
ساختار	یکپارچه و بدون ساختار	ماژولار و تابع محور	کپسوله و شیء محور
داده و رفتار	درهم‌تنیده و پراکنده	جدا از یکدیگر	یکپارچه در شیء
قابلیت استفاده مجدد	✖ خیلی ضعیف	خوب (توابع) 	عالی (کلاس‌ها و اشیاء) 
محافظت از داده	✖ وجود ندارد	ضعیف (Global State) 	(Encapsulation) قوی 
مقیاس‌پذیری	✖ غیرممکن	دشوار 	بسیار خوب 

راز اصلی OOP: اشیاء به عنوان یک تیم هماهنگ کار می‌کنند

- یک شیء به تنها یی قدرتمند نیست. قدرت واقعی در تعامل بین اشیاء نهفته است.
- درست مانند یک مدیر، یک شیء (مثلًاً Library) مسئولیت مدیریت و هماهنگی دیگر اشیاء (تیم Book‌ها) را بر عهده می‌گیرد.
- 'Library' خودش کتاب‌ها را نمی‌خواند؛ بلکه آنها را سازماندهی کرده و اطلاعاتشان را در صورت درخواست ارائه می‌دهد.

معماری سیستم کتابخانه



قانون طلایی مدیریت خوب: اصل تک مسئولیتی (SRP)

۹۹ یک مدیر عالی تضمین می‌کند که هر عضو تیم، یک و فقط یک، وظیفه اصلی داشته باشد.

- در دنیای شیءگرایی، این به معنای Single Responsibility Principle است.
- یک کلاس `Book` فقط باید نگران اطلاعات کتاب باشد، نه نحوه ذخیره آن در دیتابیس.
- یک کلاس `User` فقط باید اطلاعات کاربر را مدیریت کند، نه ارسال ایمیل.
- این اصل، کدی تمزیتر، قابل فهمتر و با قابلیت نگهداری بسیار بالاتر ایجاد می‌کند.



تابع در برابر متدها: ابزار عمومی یا رفتار تخصصی؟

هر دو کار انجام می‌دهند، اما فلسفه‌شان کاملاً متفاوت است.



تابع (Function)

`Math.sqrt(25.0)`

آنالوژی

یک آچار در جعبه ابزار عمومی. این آچار به هیچ ماشین خاصی تعلق ندارد و هر کسی می‌تواند از آن استفاده کند.

ماهیت

یک ابزار کمکی (Utility) و بدون حالت (Stateless) است. به همین دلیل `static` تعریف می‌شود.



متدها (Method)

`myAccount.getBalance()`

آنالوژی

پرسیدن از مدیر حساب شما درباره موجودی‌تان. این یک رفتار تخصصی است که فقط متعلق به آن مدیر حساب (شیء myAccount) است.

ماهیت

یک رفتار (Behavior) است که به حالت (داده‌های) یک شیء خاص وابسته است.

نوبت شماست: اعضای تیم را شناسایی کنید

تمرین: شناسایی موجودیت‌ها

> یک دانشجو برای ثبت‌نام در یک دوره آموزشی، به وب‌سایت دانشگاه مراجعه می‌کند. هر دوره توسط یک استاد ارائه می‌شود و دارای یک گد منحصر به فرد است. دانشجو می‌تواند چندین دوره را انتخاب کند.

در سناریوی بالا، «اسم‌ها» یا موجودیت‌های اصلی که کاندیدای تبدیل شدن به کلاس هستند کدامند؟ (چه کسانی بازیگران اصلی این داستان هستند؟)

پاسخ‌ها:

Student •

Course •

Professor •

(System یا به سادگی UniversityWebsite •

چالش نهایی: تیم خود را بسازید و وظایفشان را مشخص کنید

تمرین: طراحی سیستم کتابخانه

یک سیستم ساده شی‌عگرا برای کتابخانه طراحی کنید. شما به دو کلاس (دو متخصص) نیاز دارید:

1. متخصص کتاب (Book):

- دانش: title, author :(Fields)
- توانایی: displayInfo() (Methods)

2. مدیر کتابخانه (Library):

- دانش: آرایه‌ای از کتاب‌ها (Book[] books) :(Fields)
- توانایی: addBook(Book b), findBookByTitle(String title) (Methods)

سعی کنید ساختار کلی این دو کلاس را روی کاغذ یا در ویرایشگر کد خود بنویسید.

پارادایم جدید: دنیای شما یک تیم است، نه یک لیست کار

از این پس، به جای پرسیدن «چگونه این کار را انجام دهم؟»
بپرسید «چه کسی باید این کار را انجام دهد؟»



پنهان‌سازی پیچیدگی: به تیم خود اعتماد کنید. شما به عنوان مدیر، نیازی به دانستن جزئیات کار هر متخصص ندارید.



کپسوله‌سازی: هر شیء یک متخصص است که دانش و توانایی خود را در یک بسته امن نگهداری می‌کند.



تمرکز بر موجودیت‌ها: به جای افعال (تواضع)، روی اسم‌ها (اشیاء) و مسئولیت‌هایشان تمرکز کنید.

شما دیگر فقط یک کدنویس نیستید. شما یک معمار و مدیر سیستم‌های هوشمند هستید.