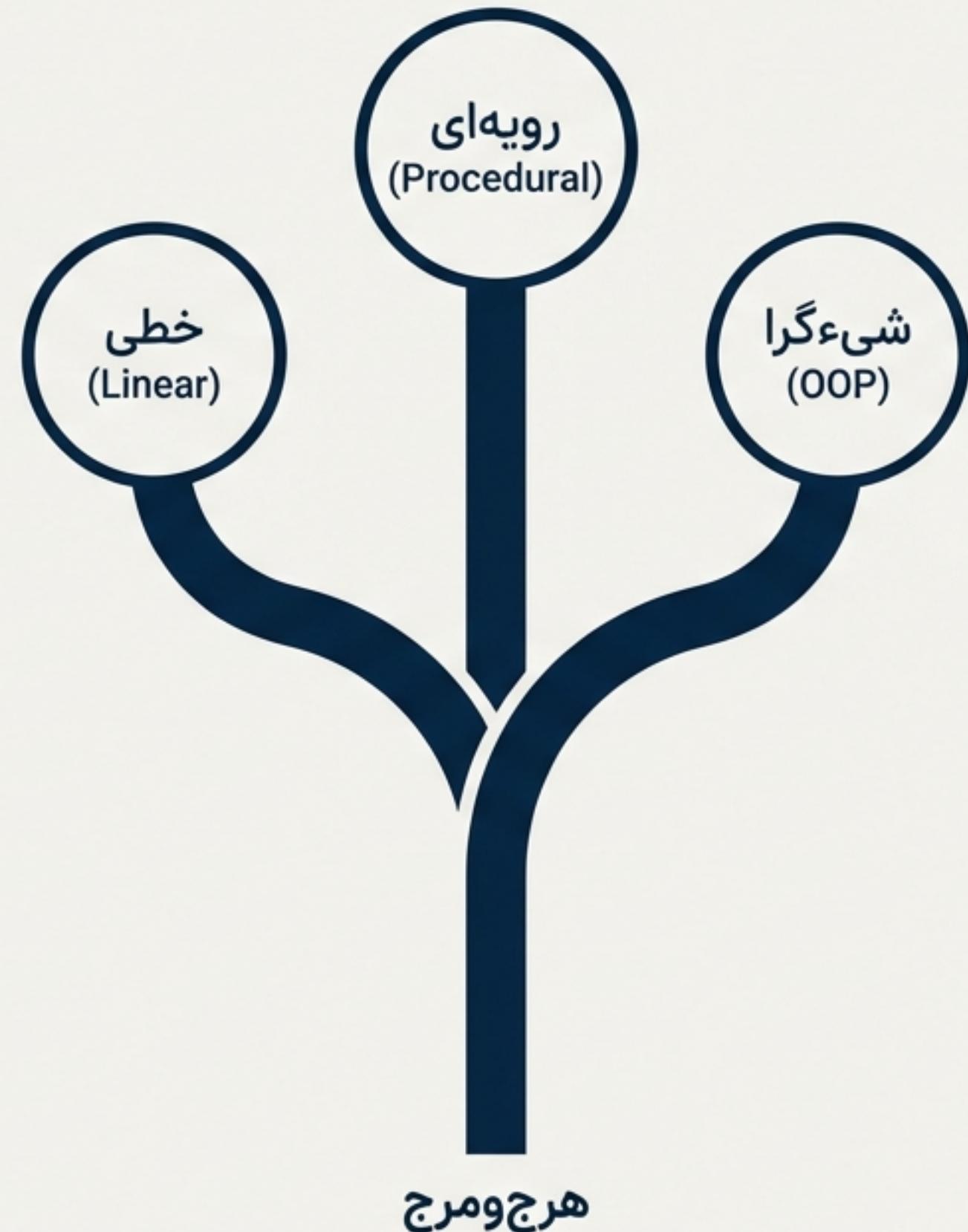


بخش ا: از پارادایم تا بایت کد — فلسفه و معماری جاوا

تهیه شده توسط: سید سجاد پیراهش

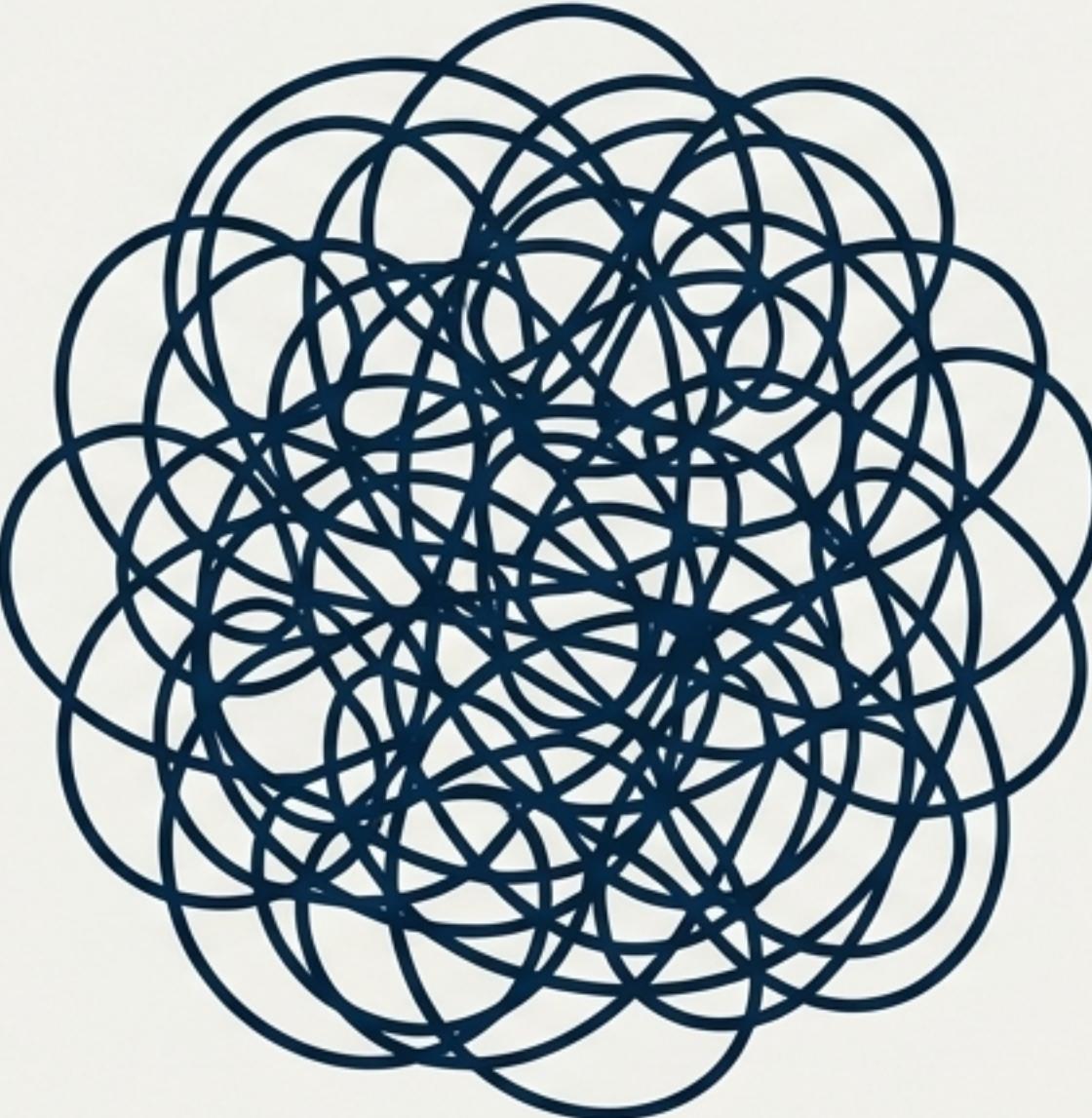
سفری از هرجومنج به ساختار: پارادایم برنامه‌نویسی چیست؟

- یک پارادایم برنامه‌نویسی (Programming Paradigm)، یک سبک، فلسفه یا روش تفکر برای ساختاردهی به کد و حل مسائل است.
- این پارادایم‌ها صرفاً «قانون» نیستند؛ بلکه پاسخ‌هایی هوشمندانه به چالش‌های نسل‌های قبلی خود هستند.
- در این بخش، سیر تکامل این تفکر را دنبال می‌کنیم تا بفهمیم چرا شی‌عگرایی یک انتخاب نیست، بلکه یک ضرورت برای ساخت سیستم‌های پیچیده است.



نقطه آغاز: پارادایم خطی و کابوس «کد اسپاگتی»

- در این رویکرد، برنامه یک بلوک عظیم و یکپارچه از دستورات است که از بالا به پایین اجرا می‌شود.
- **تشبیه:** یک کتاب طولانی بدون هیچ فصل، پاراگراف یا عنوانی.
- **مشکلات بنیادین:**
- **کد اسپاگتی (Spaghetti Code):** با بزرگ شدن برنامه، جریان اجرا به کلافی درهم‌پیچیده از دستورات `goto` تبدیل می‌شد که درک و دنبال کردن آن غیرممکن بود.
- **عدم استفاده مجدد (No Reusability):** هر کار تکراری نیازمند کپی-پیست کردن کد بود.
- **کابوس نگهداری:** تغییر یک بخش می‌توانست به صورت فاجعه‌بار و غیرمنتظره، بخش‌های نامرتب دیگر را دچار خطا کند (یکند (اثر پروانه‌ای)).



عدم استفاده مجدد



نگهداری سخت

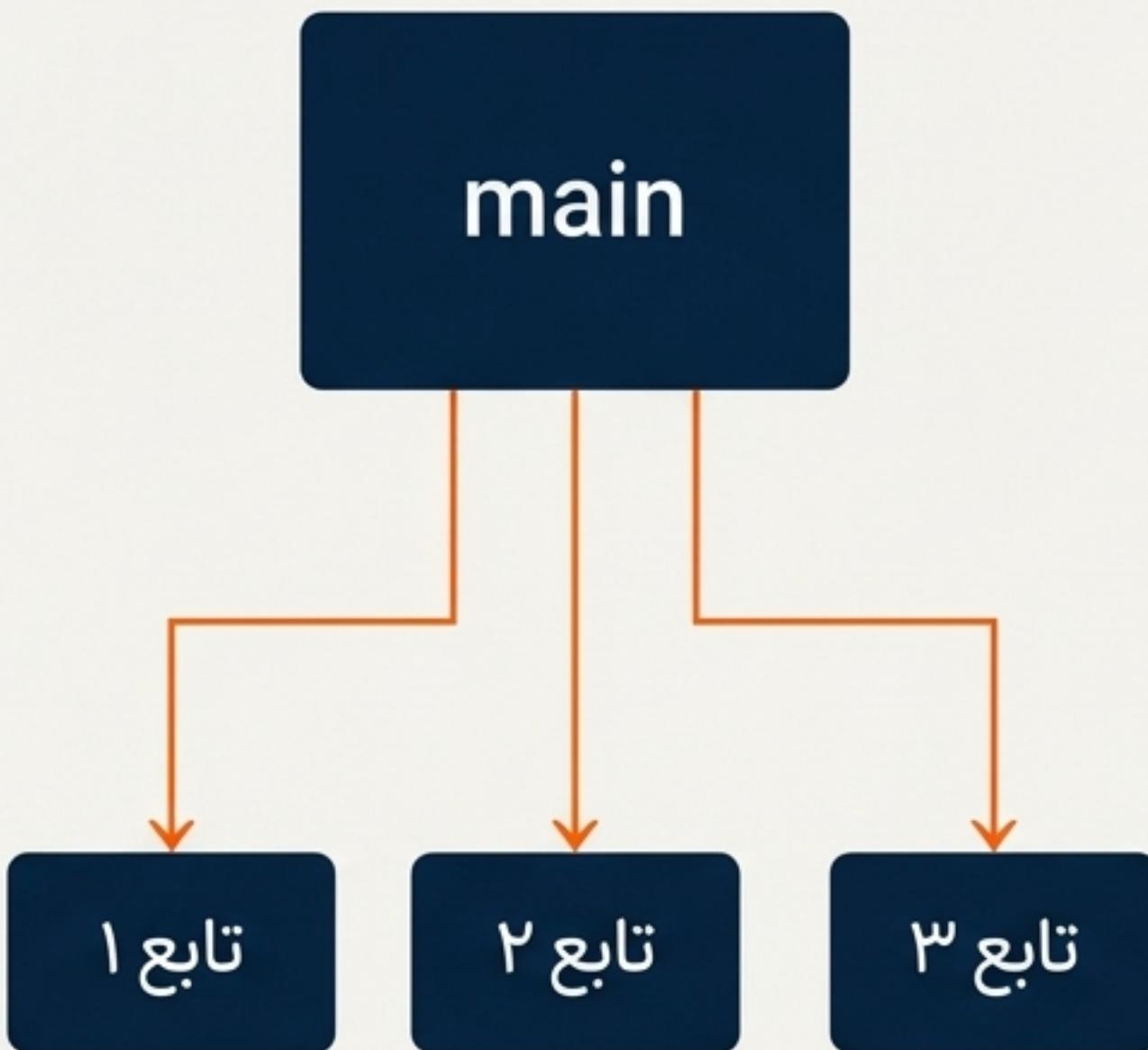
انقلاب رویه‌ای: استراتژی «تفرقه بینداز و حکومت کن»



این پارادایم برای حل فاجعه «کد اسپاگتی» ظهرور کرد.

* **ایده محوری:** شکستن یک وظیفه بزرگ به زیروظیفه‌های کوچک‌تر و قابل مدیریت به نام **نام رویه (Procedure)** یا **تابع (Function)**.

* **ویژگی کلیدی:** جدایی کامل **داده‌ها (Data)** از رفتارها (Behaviors/Functions). توابع روی داده‌های مستقل کار می‌کنند.



پاشنه آشیل: اثر پروانه‌ای در داده‌های اشتراکی

- بزرگترین نقطه قوت پارادایم رویه‌ای (جدایی داده و رفتار) به ضعف اصلی آن تبدیل شد.
- مشکل: هیچ کنترلی روی اینکه «کدام تابع» می‌تواند «کدام داده» را تغییر دهد، وجود نداشت.
- متغیرهای سراسری (Global Variables) مانند یک انبار عمومی بدون نگهبان بودند. هر تابعی می‌توانست داده‌ها را تغییر دهد و باعث ایجاد باگ‌های پنهان و غیرقابل پیش‌بینی در توابع دیگر شود.



انقلاب شیءگرایی: اتحاد داده و رفتار در «اشیاء»



OOP برای حل مشکل اساسی پارادایم رویه‌ای (عدم حفاظت از داده‌ها) به وجود آمد.

ایده محوری: به جای جدا کردن داده و رفتار، آنها را در یک واحد یکپارچه و «Object» نام «شیء» محافظت شده به نام «شیء» (Object) بسته‌بندی می‌کنیم.



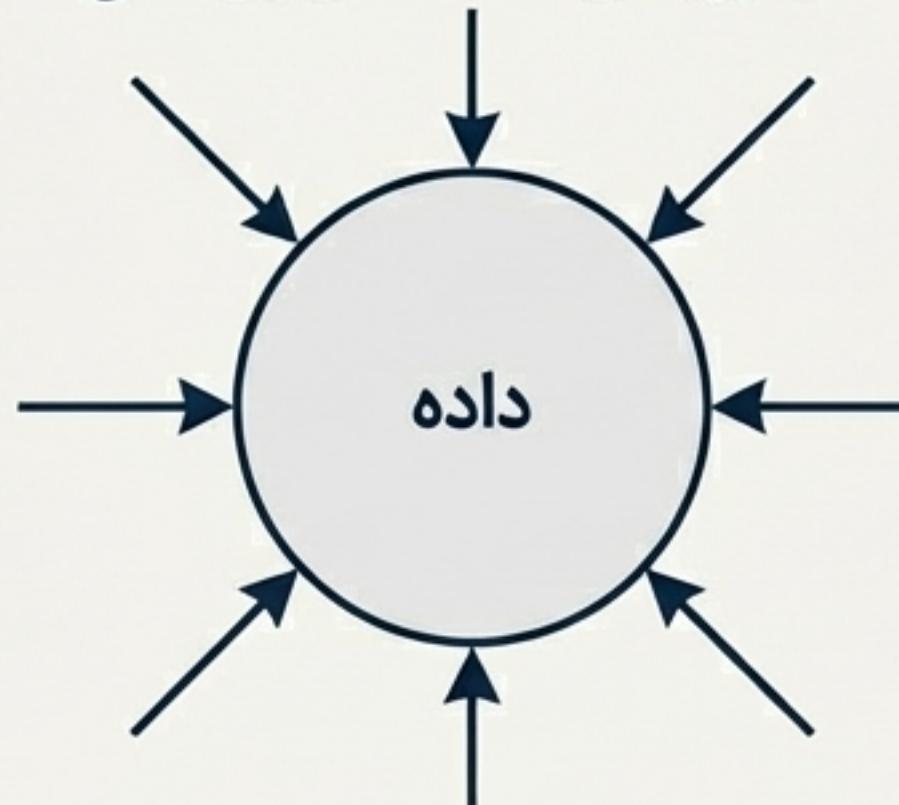
شیء (Object): یک نمونه واقعی و زنده

- کلاس (Class): نقشه یک دانشجو
 - شیء (Object): شیء دانشجوی علی و شیء دانشجوی سارا

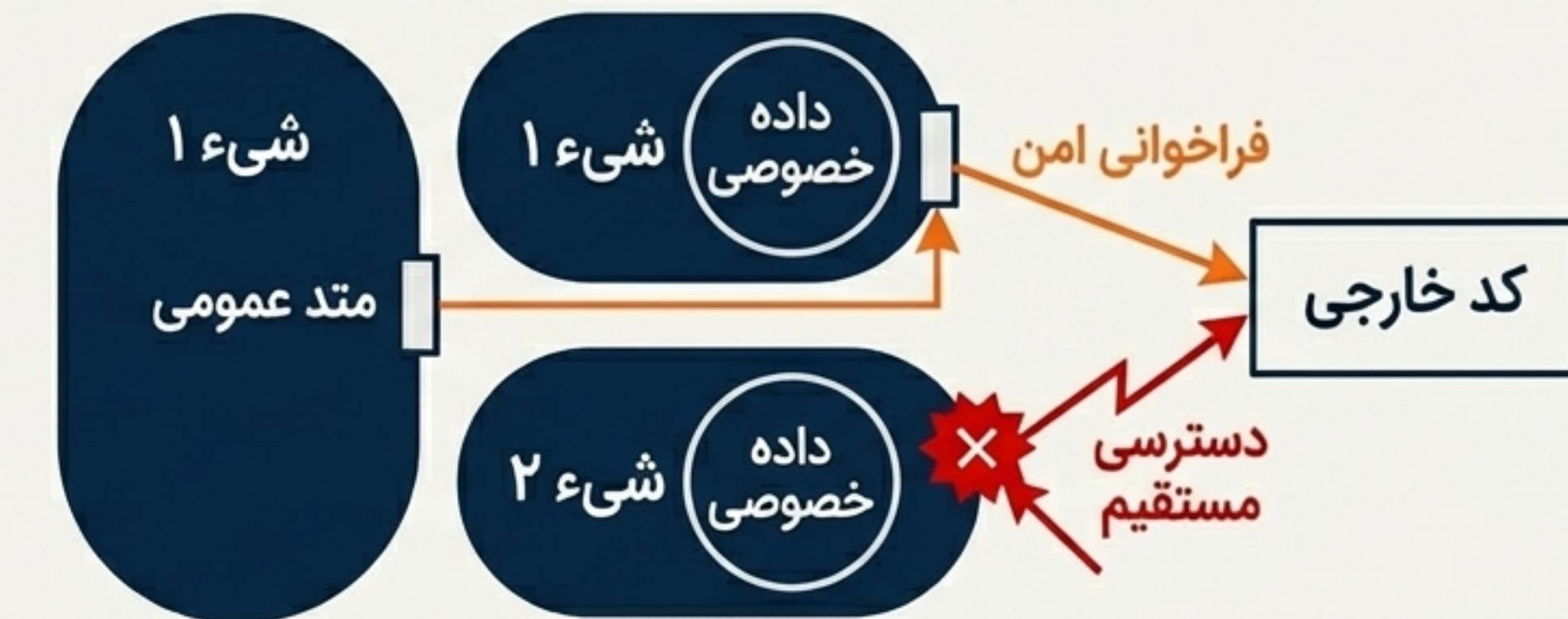
کپسوله‌سازی (Encapsulation): نگهبان قدرتمند داده‌ها

- این قلب تپنده OOP و راه حل مستقیم «اثر پروانه‌ای» است.
- مکانیزم: داده‌های یک شیء (fields) (خصوصی) تعریف شده و از دنیای بیرون مخفی می‌شوند.
- دسترسی به این داده‌ها فقط از طریق متدهای public (عمومی) همان شیء امکان‌پذیر است. این متدها مانند یک «نگهبان» عمل کرده و قبل از هر تغییری، قوانین و اعتبارسنجی‌های لازم را اعمال می‌کنند.

مدل رویه‌ای: داده‌های بی‌دفاع



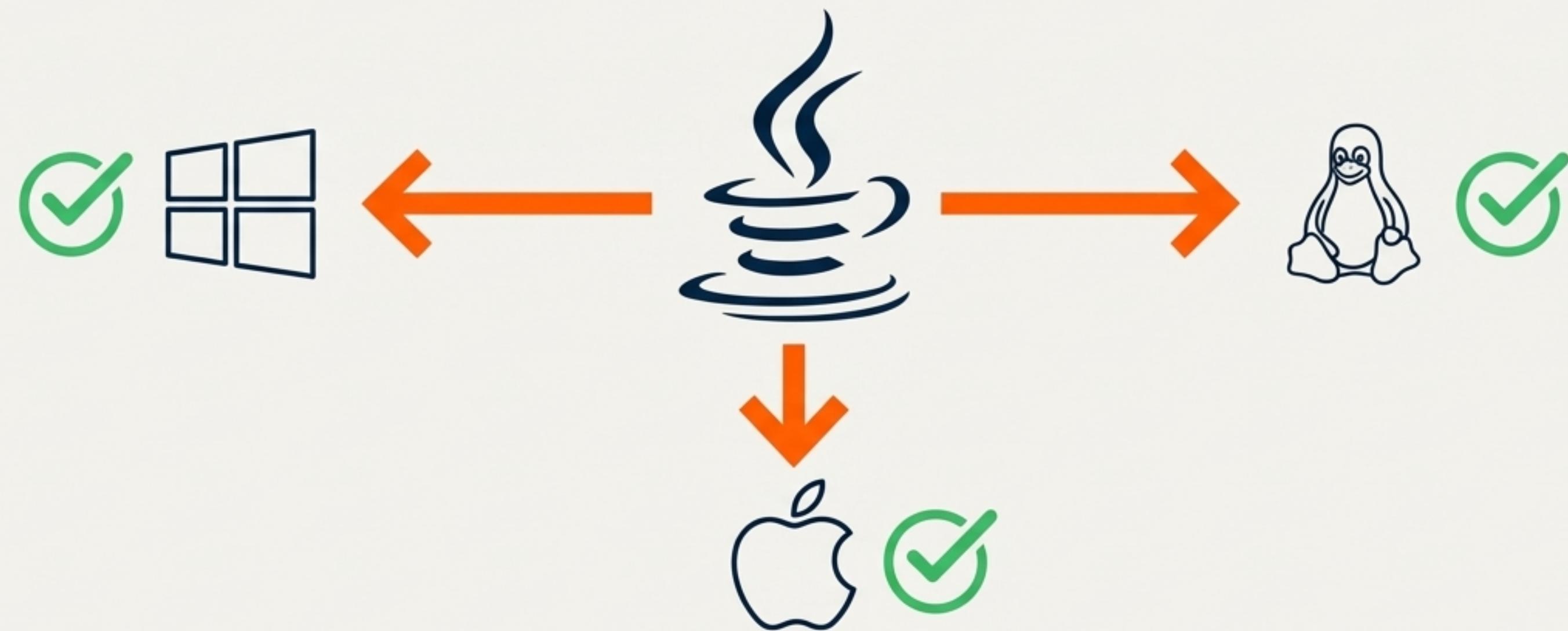
مدل شیء‌گرا: داده‌های محافظت‌شده



مقایسه نهایی پارادایم‌ها: از یکپارچه تا شیء‌محور

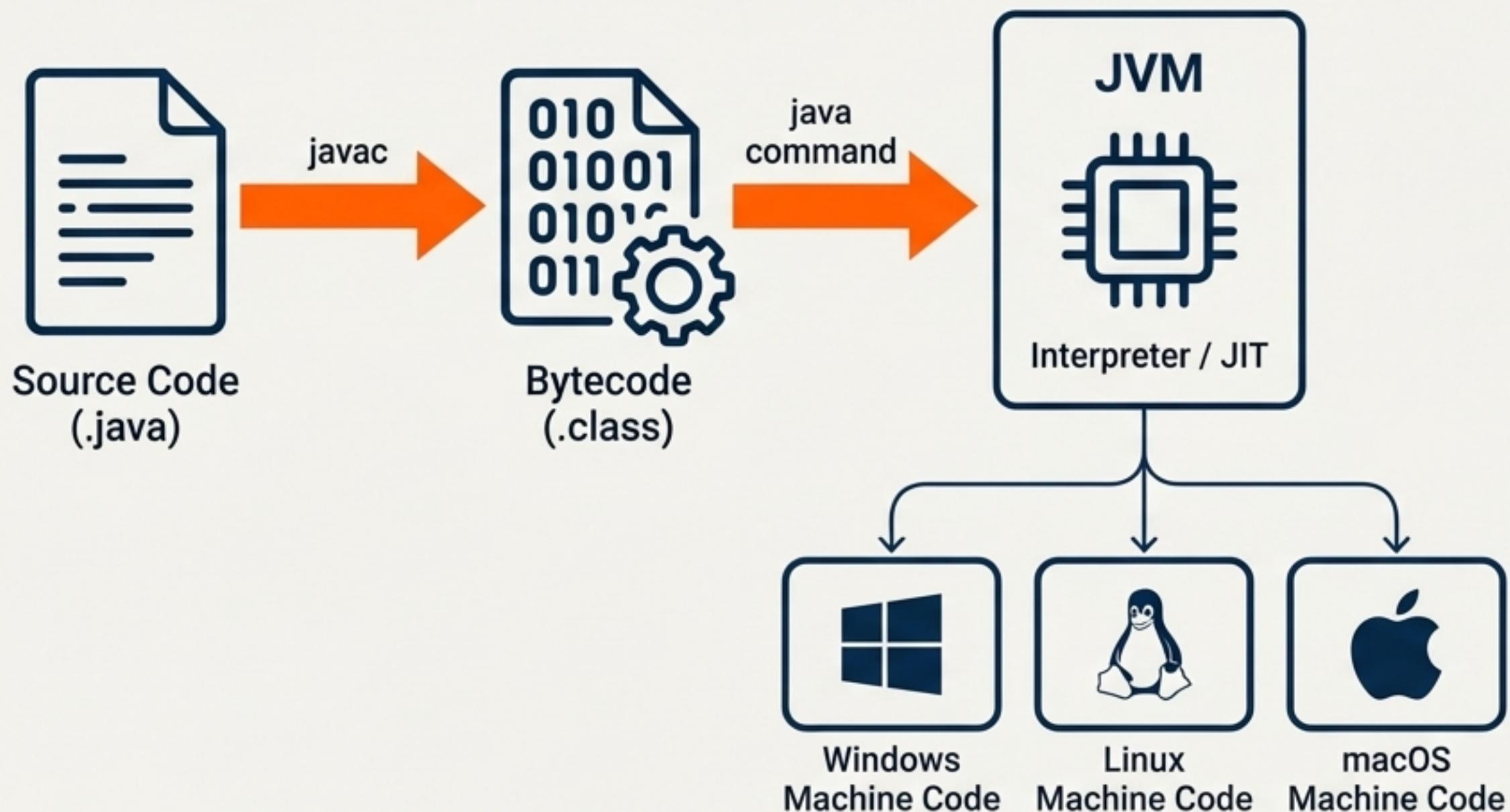
	Linear (خطی)	Procedural (رویه‌ای)	OOP (شیء‌گرا)	معیار
ساختار	یکپارچه	تابع‌محور	شیء‌محور	
داده و رفتار	درهم	جدا	یکپارچه	
قابلیت استفاده مجدد	ضعیف	خوب	عالی	
محافظت از داده	هیچ	ضعیف	قوی	
پیچیدگی مدیریت	بسیار سخت	متوسط	ساده	
مقیاس‌پذیری		!		
مناسب برای	اسکریپت‌های کوچک	برنامه‌های متوسط	سیستم‌های بزرگ	
زبان‌های نمونه	BASIC قدیمی	C, Pascal	Java, C++, Python	

معماری جاوا: فلسفه «یک بار بنویس، همه‌جا اجرا کن»



- * شعار جاوا (**Write Once, Run Anywhere**) راه حلی برای یک مشکل بزرگ در دهه ۹۰ بود: برنامه‌ای که برای ویندوز کامپایل می‌شد، روی مک یا لینوکس اجرا نمی‌شد.
- * جاوا این مشکل را با معرفی یک لایه میانی هوشمند به نام **ماشین مجازی جاوا (Java Virtual Machine - JVM)** حل کرد.
- * JVM مانند یک مترجم جهانی عمل می‌کند که بین کد شما و سیستم‌عامل‌های مختلف قرار می‌گیرد و استقلال کامل از پلتفرم را تضمین می‌کند.

کالبدشکافی فرآیند: از سورس کد (java.) تا اجرا



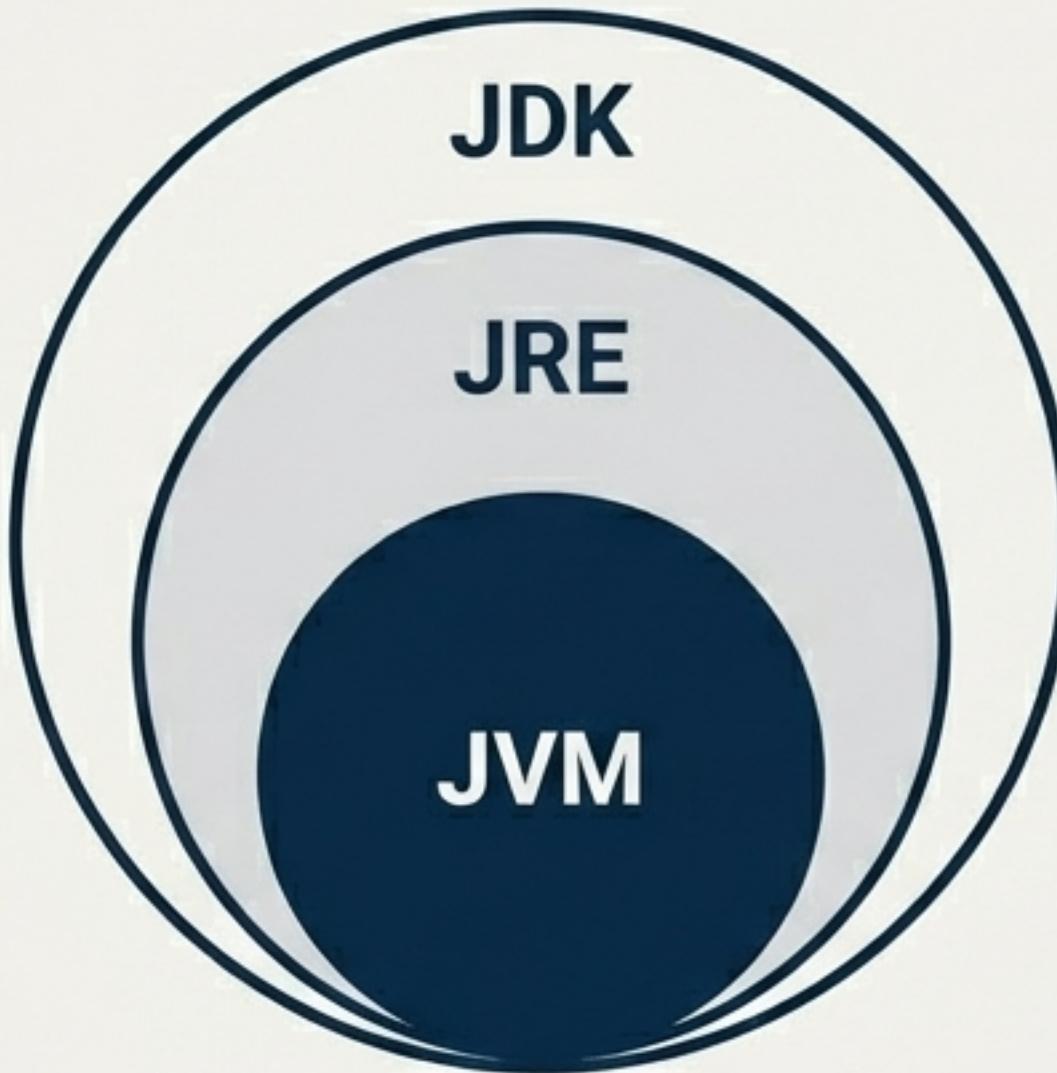
1. **.java (کد منبع):** کدی که شما می‌نویسید و برای انسان قابل فهم است.

2. **javac (کامپایلر):** کد شما را به یک زبان میانی جهانی به نام **بایت‌کد (Bytecode)** ترجمه می‌کند.

3. **.class (بایت‌کد):** خروجی کامپایلر که برای یک پردازنده مجازی (JVM) طراحی شده، نه یک پردازنده فیزیکی.

4. **JVM (ماشین مجازی):** این نرم‌افزار بایت‌کد را گرفته و آن را به کد ماشین مخصوص همان سیستم‌عامل تبدیل و اجرا می‌کند.

تثییث مقدس جاوا: JDK vs JRE vs JVM



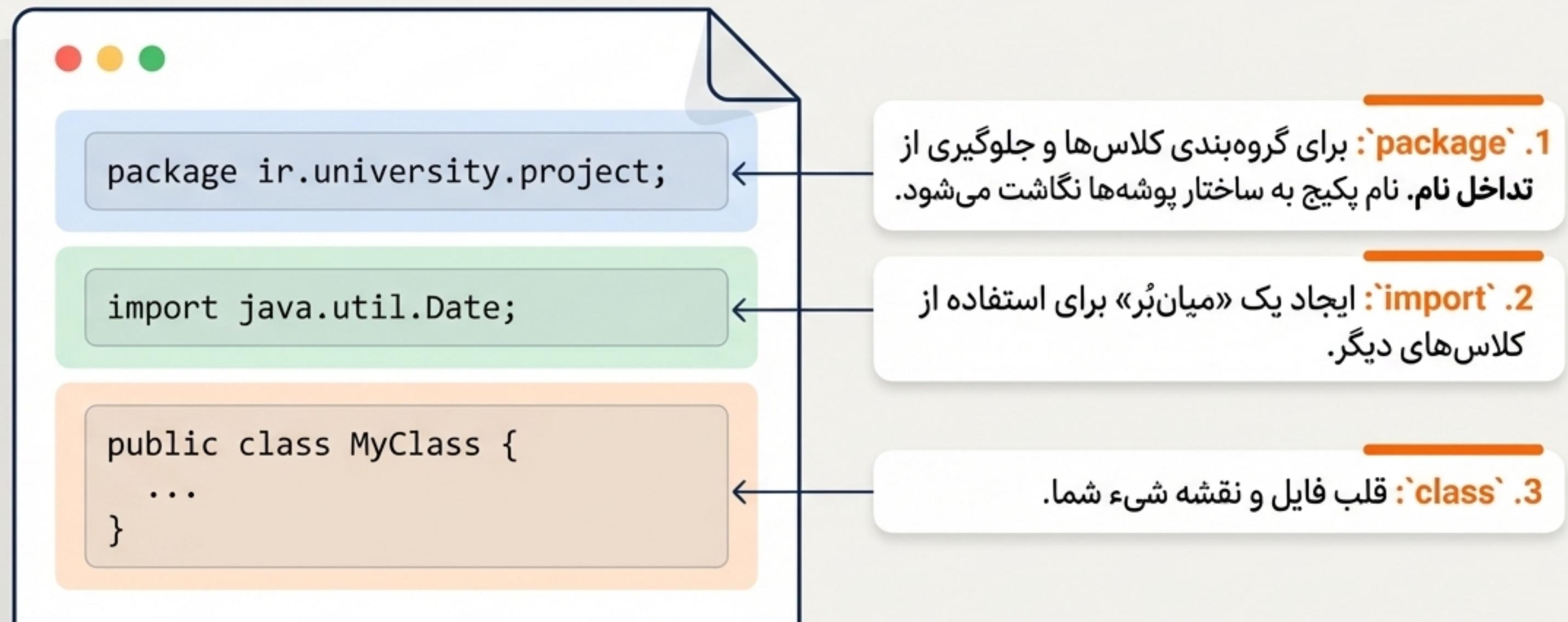
JDK: کیت توسعه (برای برنامهنویسان)

JRE: محیط اجرا (برای کاربران)

JVM: هسته اصلی

کاربرد	مخاطب	شامل چیست؟	مفهوم
اجرای بایت‌کد	-	موتور اجرا	JVM
اجرای برنامه	کاربر نهایی	JVM + کتابخانه‌ها	JRE
توسعه و اجرا	برنامه‌نویس	JRE + ابزارهای توسعه	JDK

آناتومی یک فایل جاوا: قوانین پکیج، ایمپورت و کلاس



قانون طلایی: در هر فایل `.java`، حداقل یک کلاس `public` می‌تواند باشد و نام فایل باید دقیقاً با نام آن کلاس یکسان باشد.

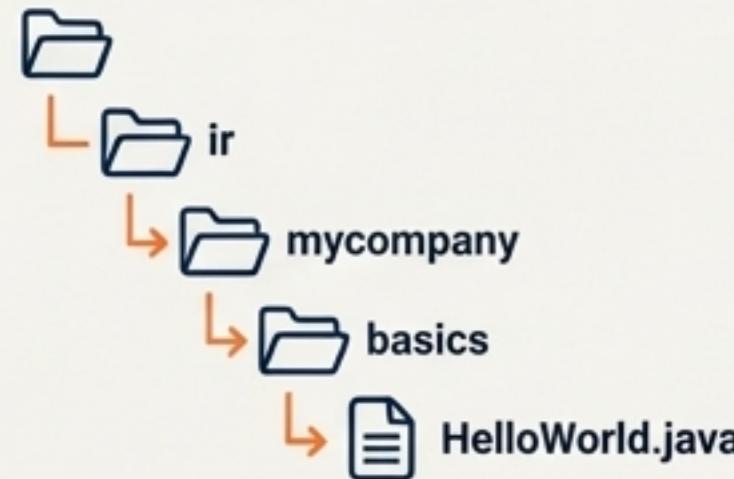
نمونه عملی: از کد تا خروجی در یک «Hello World» کامل

کد منبع

```
1 package ir.mycompany.basics;  
2  
3 import java.util.Date;  
4  
5 public class HelloWorld {  
6     public static void main(String[] args) {  
7         System.out.println("Hello, World!");  
8         System.out.println("Today is: " + new Date());  
9         if (args.length > 0) {  
10             System.out.println("Hello, " + args[0] + "!");  
11         }  
12     }  
13 }
```

ساختار و اجرا

** ساختار پوشه مورد نیاز



کامپایل و اجرا

```
# Move to the root directory above 'ir'  
javac ir/mycompany/basics/HelloWorld.java  
  
# Run from the same root directory  
java ir.mycompany.basics.HelloWorld  
  
# Run with an argument  
java ir.mycompany.basics.HelloWorld Sajjad
```

خروجی

```
Hello, World!  
Today is: [Current Date and Time]  
  
Hello, World!  
Today is: [Current Date and Time]  
Hello, Sajjad!
```

نوبت شماست: مهارت خود را بسازید



تمرین ۱: اولین برنامه

یک فایل `Welcome.java` در پکیج `ir.training.day1`، training باشید که نام شما و تاریخ و زمان فعلی را چاپ کند.



تمرین ۲: ماشین حساب ساده

فایلی به نام `Calculator.java` بسازید که یک کلاس `Calculator` با یک متدهای `main` داشته باشد که $5 + 10$ را محاسبه و چاپ کند.



تمرین ۳: آرگومان‌های خط فرمان

برنامه‌ای بنویسید که دو عدد را از آرگومان‌های خط فرمان بگیرد، جمع آنها را چاپ کند، و در صورت عدم ارائه آرگومان، پیام خطا بدهد.



تمرین ۴: ساختار پکیج

ساختار پوشه `com/myproject/utils` را بسازید و یک کلاس ساده به نام `Helper` در آن تعریف کنید.

خلاصه بخش اول و نگاهی به آینده

- تکامل پارادایم‌ها: از Linear و هرجومنج، به پرودرود و در نهایت به ساختار قدرتمند OOP رسیدیم. ✓
- فلسفه OOP: اتحاد داده و رفتار در اشیاء، با قدرت کپسوله‌سازی برای حفاظت از داده‌ها. ✓
- معماری جاوا: فرآیند `javac` → `.java` → JVM. را درک کردیم. ✓
- شعار WORA: فهمیدیم چگونه JVM استقلال از پلتفرم را ممکن می‌سازد. ✓
- تثییث جاوا: تفاوت و رابطه JDK ⊃ JRE ⊃ JVM را یاد گرفتیم. ✓
- آناتومی فایل: با قوانین `package`, `import`, و `.class`. آشنا شدیم. ✓

در بخش بعدی: به سراغ آجرهای سازنده اشیاء می‌رویم: **انواع داده، تفاوت حیاتی حافظه و Heap، و مفهوم فیلد**ها.

