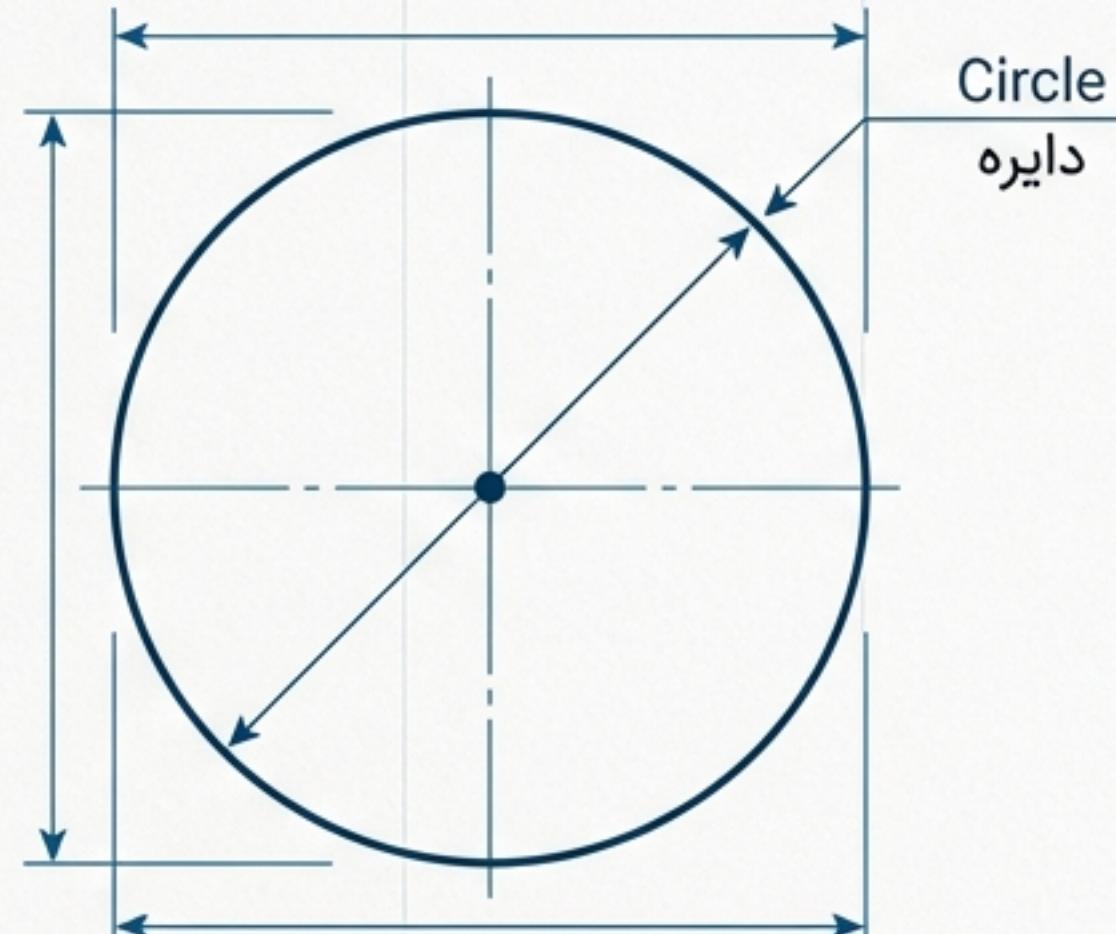


بخش ۱۳: کلاس‌های انتزاعی (Abstract Classes)

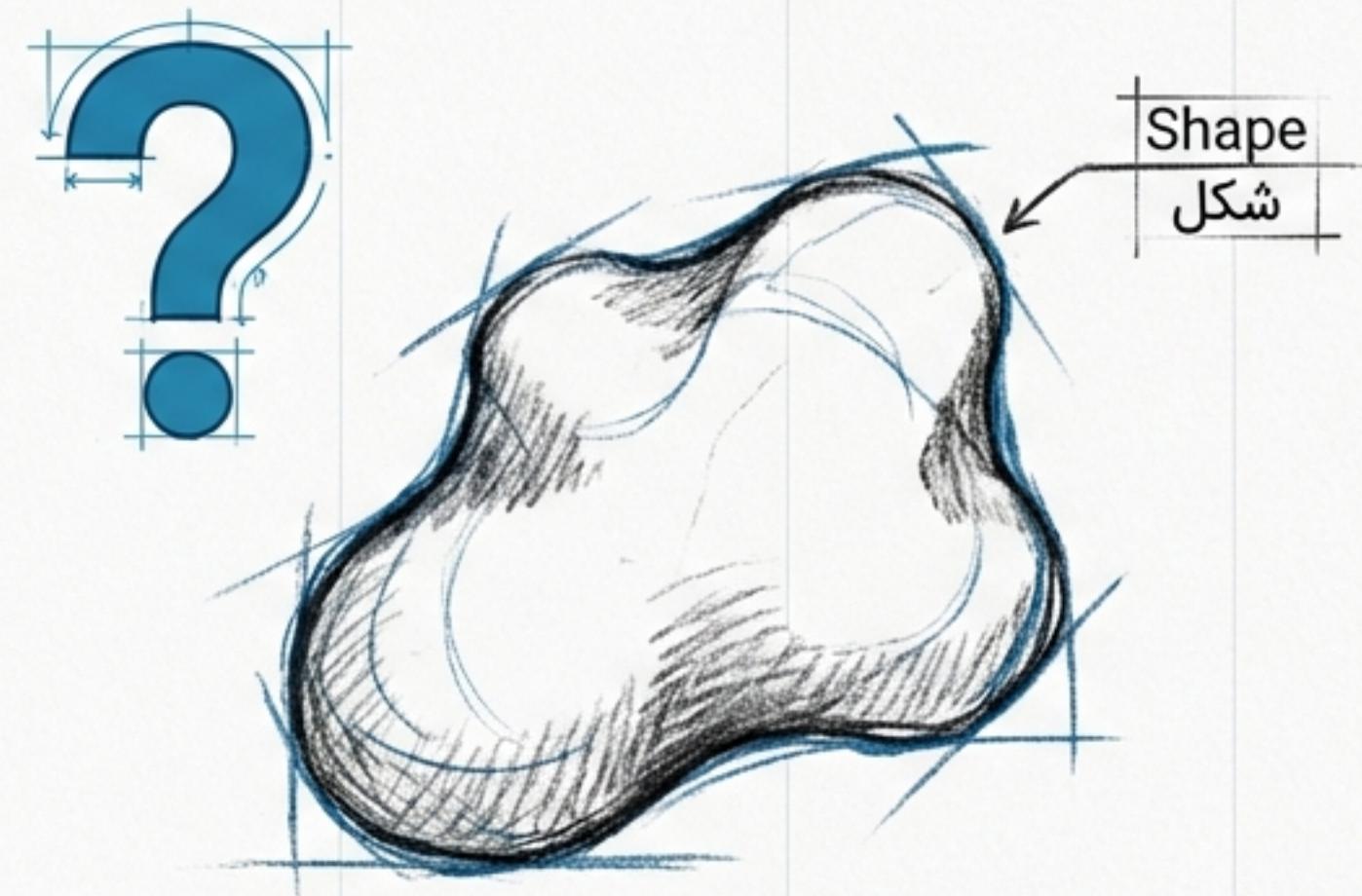
تعریف قرارداد بدون پیاده‌سازی

تهیه شده توسط: سید سجاد پیراھش

یک سوال بنیادین: آیا می‌توان 'ایده' را ساخت؟



Blueprint of a tangible object

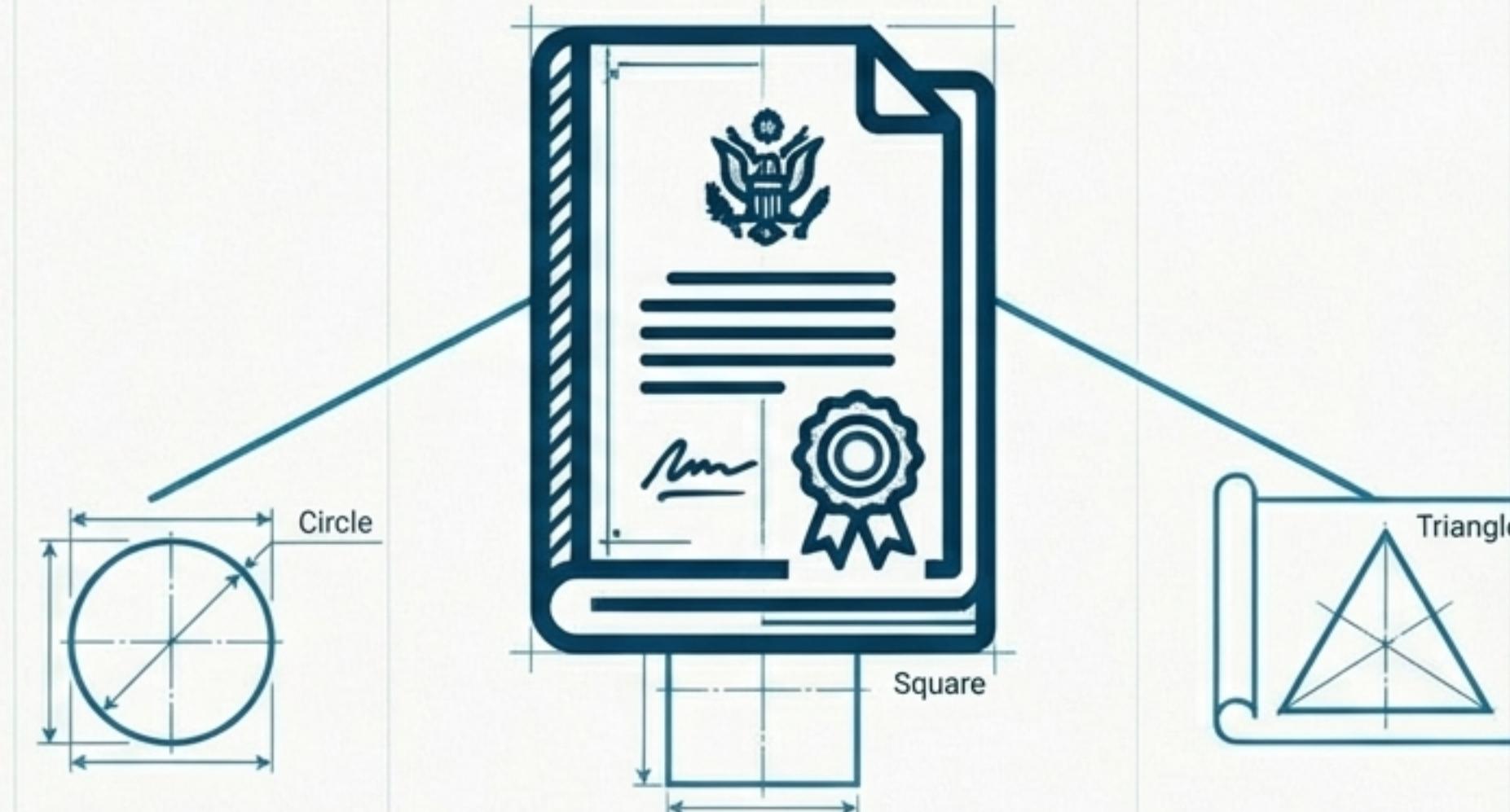


Conceptual sketch of an abstract idea

به کلاس `Shape` (شکل) فکر کنید. ما می‌توانیم یک `Shape` (مربع) یا یک `Circle` (دایره) بسازیم. این‌ها موجودیت‌های واقعی هستند. اما خود `Shape` یک مفهوم کلی و انتزاعی است. تلاش برای ساختن یک شیء از یک مفهوم خالص (new Shape()) در طراحی، یک خطای منطقی است. ما به ابزاری نیاز داریم تا بتوانیم *ایده* را تعریف کنیم، بدون آنکه اجزاء ساختنش را بدهیم.



معرفی کلاس انتزاعی: قانون اساسی سلسله مراتب کد شما



- یک کلاس انتزاعی، طرحی برای *سایر طرح‌ها* است. این کلاس عمدتاً ناتمام است.

• **تشبیه قدرتمند:** کلاس انتزاعی مانند یک قانون اساسی عمل می‌کند. قوانین بنیادین را تعیین می‌کند (کارهایی که تمام زیرکلاس‌ها باید انجام دهند) و منابع مشترک را فراهم می‌آورد (کدی که تمام زیرکلاس‌ها می‌توانند استفاده کنند). اما خودش یک 'شهروند' نیست. شما نمی‌توانید از آن یک نمونه ([new](#)) بسازید.

آناتومی قدرت: متدهای انضمامی در برابر متدهای انتزاعی

کارهای انجام شده (متد انضمامی)

این‌ها کدهای مشترک و آماده استفاده هستند. مانند خدمات عمومی که برای همه فرزندان فراهم شده است.

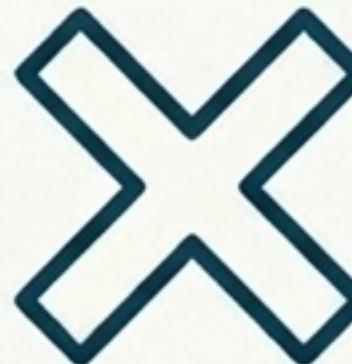
```
// Shared implementation for all shapes
public void display() {
    System.out.println("Displaying a shape.");
}
```

کارهای الزامی (متد انتزاعی)

این‌ها قرارداد هستند. یک "لیست وظایف" که به فرزندان خود می‌دهید. یک قول بدون پیاده‌سازی. فقط امضا، بدون بدنه `}`.

```
// A contract. The child MUST implement this.
public abstract double getArea();
```

قوانين بازی: چگونه از این قدرت استفاده کنیم



قانون اول: هرگز با **new**

شما نمی‌توانید از یک کلاس انتزاعی شیء بسازید. این یک مفهوم ناتمام است.
تپیکشمند.



قانون دوم: فرزندان باید قرارداد را امضا کنند.

هر کلاس فرزندی باید تمام متدهای انتزاعی والد را پیاده‌سازی (Override) کند.



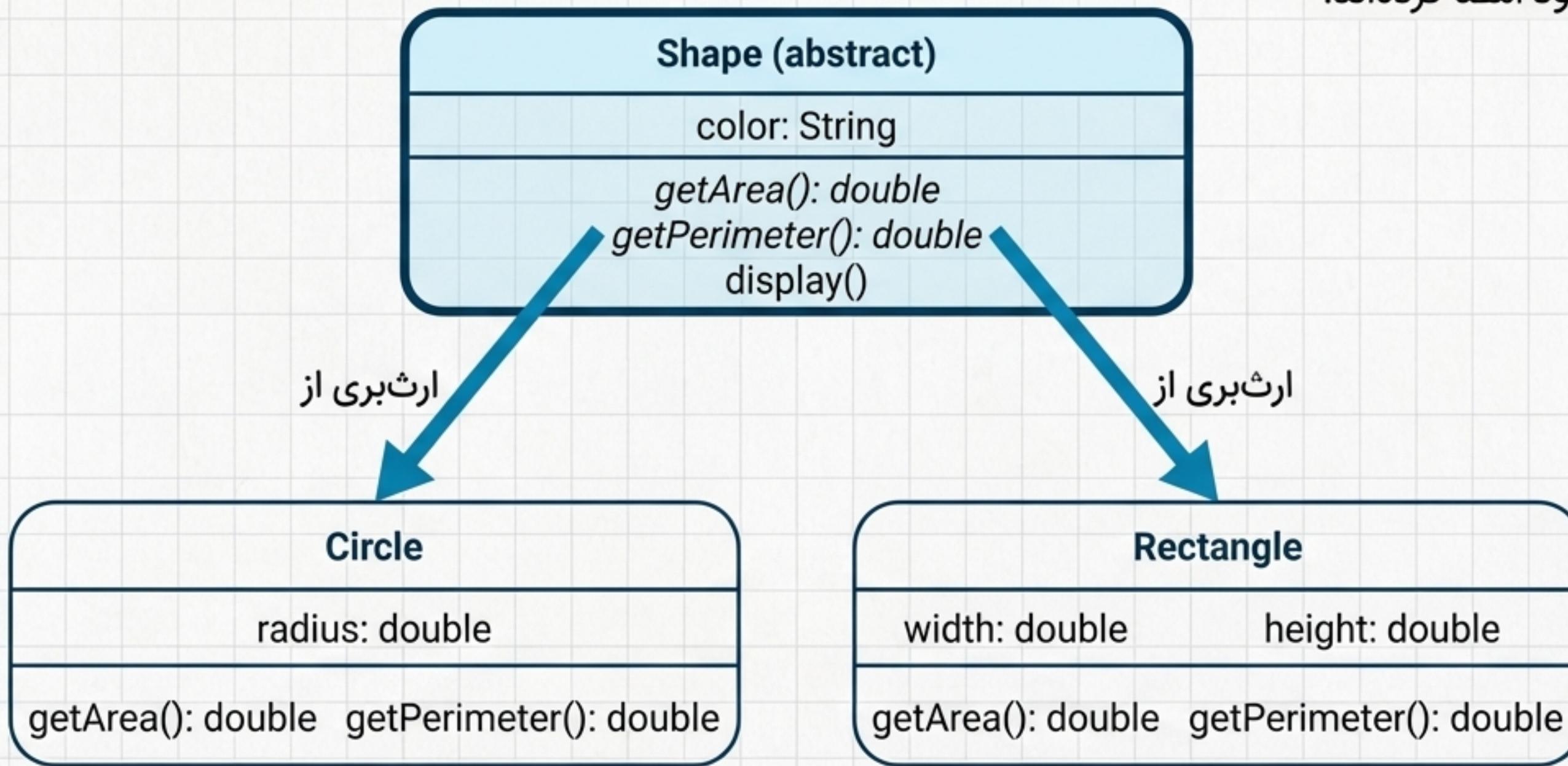
قانون سوم: یا خودشان هم انتزاعی شوند.

اگر یک کلاس فرزند حتی یکی از متدهای انتزاعی را پیاده‌سازی نکند، باید خودش خودش نیز با کلمه کلیدی **abstract** تعریف شود.



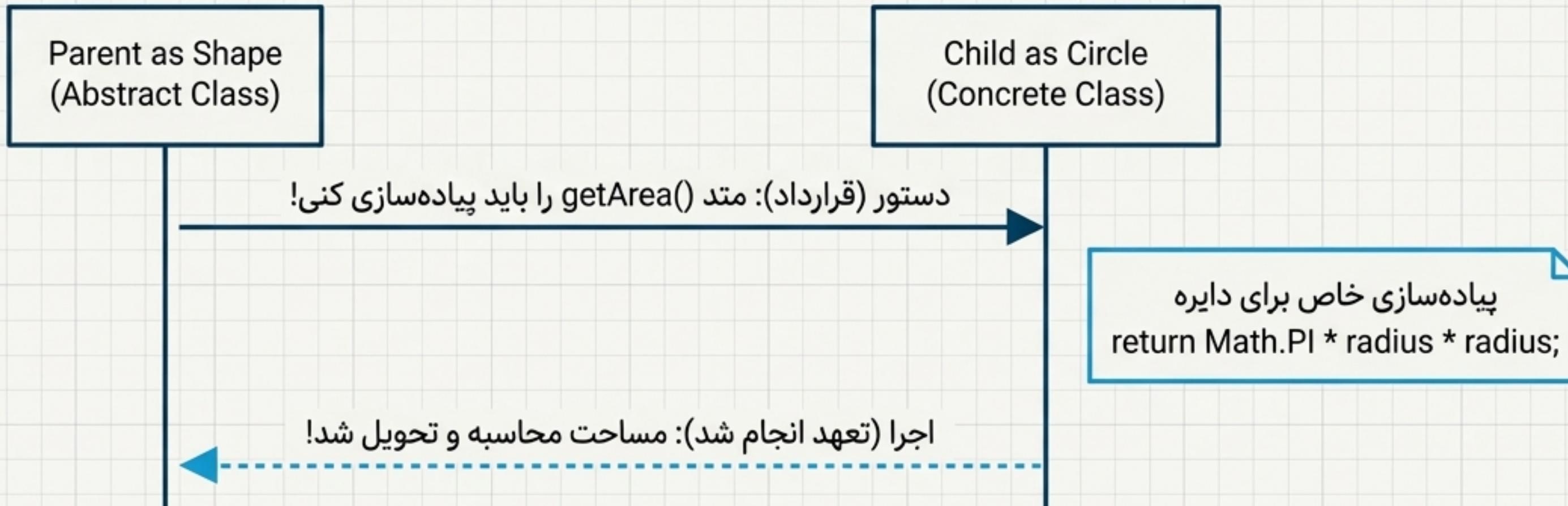
نقشه سلسه مراتب: طرح اصلی و پیاده سازی ها

در این نقشه، کلاس Shape انتزاعی و قانونگذار است. متدهای `getPerimeter` و `getArea` در آن به صورت ایتالیک مشخص شده اند تا نشان دهنده "قرارداد" بودن آنها باشد. فرزندانی هستند که این قرارداد را با پیاده سازی های مشخص خود امضا کرده اند.



جريان قرارداد: والد دستور می‌دهد، فرزند اجرا می‌کند

این دیاگرام به سادگی نشان می‌دهد که کلاس انتزاعی `Shape` چگونه یک "دستور" یا "تعهد" (متده `getArea`) را به کلاس فرزند `Circle` تحمیل می‌کند. `Circle` چاپرهای جز اطاعت و ارائه پیاده‌سازی مشخص خود ندارد.



صحنه عمل: ساخت نرم افزار نقاشی (بخش کد)

گام ۱: طراحی کلاس انتزاعی

```
// The Blueprint: Define the contract
public abstract class Shape {
    String color;
    // ... constructor ...

    // Concrete method (shared code)
    public String getColor() { return color; }

    // Abstract methods (the contract)
    public abstract double getArea();
    public abstract double getPerimeter();
}
```

گام ۲: پیاده سازی کلاس انضمامی

```
// The Implementation: Fulfill the contract
public class Circle extends Shape {
    private double radius;
    // ... constructor ...

    @Override // Signing the contract
    public double getArea() {
        return Math.PI * radius * radius;
    }

    @Override // Signing the contract
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }
}
```

قدرت واقعی: یک لیست، بی‌نهایت رفتار

این جادوی واقعی است. ما لیستی از `Shape` داریم، اما هرگز یک شیء `Shape` در آن قرار نمی‌دهیم. با این حال، می‌توانیم با تمام آشکال به صورت یکسان رفتار کنیم. در حلقه `for`، جاوا در زمان اجرا تشخیص می‌دهد که هر `S` واقعاً یک `Circle` است یا `Rectangle` و نسخه صحیح متد `getArea()` را فراخوانی می‌کند. این همان چندریختی (Polymorphism) است.

گام ۳: استفاده از قدرت چندریختی

```
گام ۳: استفاده از قدرت چندریختی //
List<Shape> shapes = new ArrayList<>();
shapes.add(new Circle(5.0));
shapes.add(new Rectangle(4.0, 6.0));

for (Shape currentShape : shapes) {
    // Java decides at runtime which getArea() to call!
    System.out.println("Area: " + currentShape.getArea());
}
```

دوئل طراحی: کلاس انضمایی در مقابل کلاس انتزاعی

Abstract Class (کلاس انتزاعی)

✗ نمی‌توان با new از آن شیء ساخت.
(خطای کامپایل)

✓ می‌تواند (و معمولاً دارد) یک یا چند متدهای انتزاعی داشته باشد.

ناتمام: یک "قالب" یا "قرارداد" برای کلاس‌های دیگر است.

✓ دارد؛ اما فقط برای فراخوانی توسط سازنده کلاس‌های فرزند (super).

فراهم کردن یک پایه مشترک و تعریف یک قرارداد برای مجموعه‌ای از کلاس‌های مرتبط.

Concrete Class (کلاس انضمایی)

✓ می‌توان با new از آن شیء ساخت.

✗ نمی‌تواند متدهای انتزاعی داشته باشد.

کامل: یک موجودیت قابل استفاده و مستقل است.

✓ دارد؛ برای ساختن شیء.

نمایش یک موجودیت واقعی و قابل استفاده در سیستم.

معیار

قابلیت نمونه‌سازی

متدهای انتزاعی

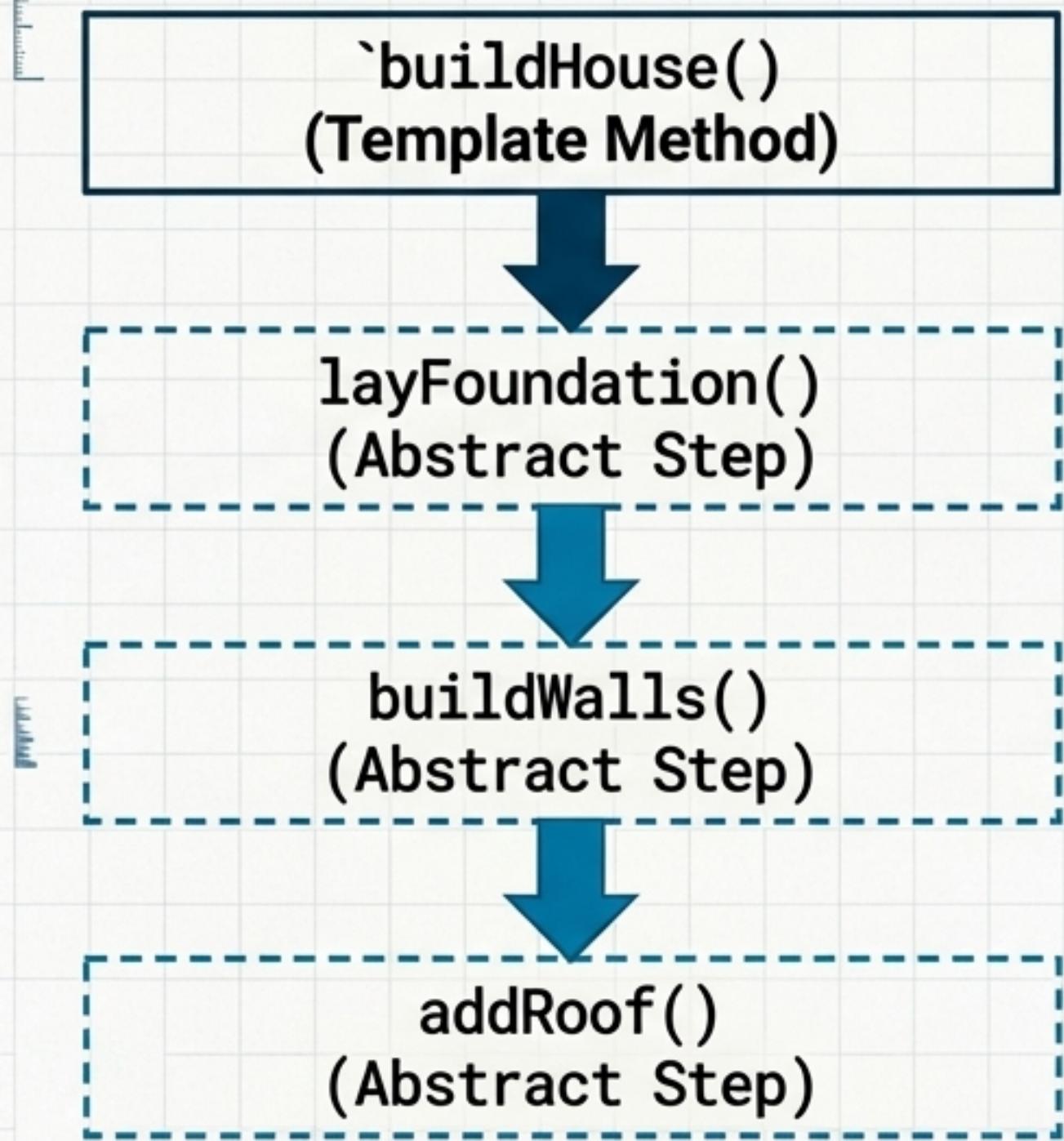
کامل بودن

سازنده

هدف اصلی



یک تکنیک استادی: الگوی متده قالب (Template Method Pattern)



کلاس‌های انتزاعی به ما اجازه می‌دهند اسکلت یک الگوریتم را در کلاس والد تعریف کنیم و پیاده‌سازی جزئیات را به فرزندان بسپاریم. متده والد (که `final` است) ترتیب مراحل را مشخص می‌کند، و فرزندان هر مرحله را پر می‌کنند.

:**Example**

تصور کنید یک متده `buildHouse()` داریم که مراحل را به ترتیب فراخوانی می‌کند:

1. `layFoundation()` (انتزاعی)
2. `buildWalls()` (انتزاعی)
3. `addRoof()` (انتزاعی)

ساختار الگوریتم در والد ثابت است، اما جزئیات هر مرحله (نوع فونداسیون، جنس دیوارها) در فرزندان مشخص می‌شود.

چالش اول: بازآفرینی سیستم کارمندان

The Problem

کلاس `Employee` زیر طراحی ضعیفی دارد.
متدهای `calculatePay` برای یک کارمند عمومی
منطقی نیست و یک مقدار جادویی (-1) یا
(-1) برمی‌گرداند.

```
// Before: Flawed Design
public class Employee {
    private String name;
    // ...
    public double calculatePay() {
        // ??? What is the pay for a generic employee?
        return -1; // Bad practice!
    }
}
```

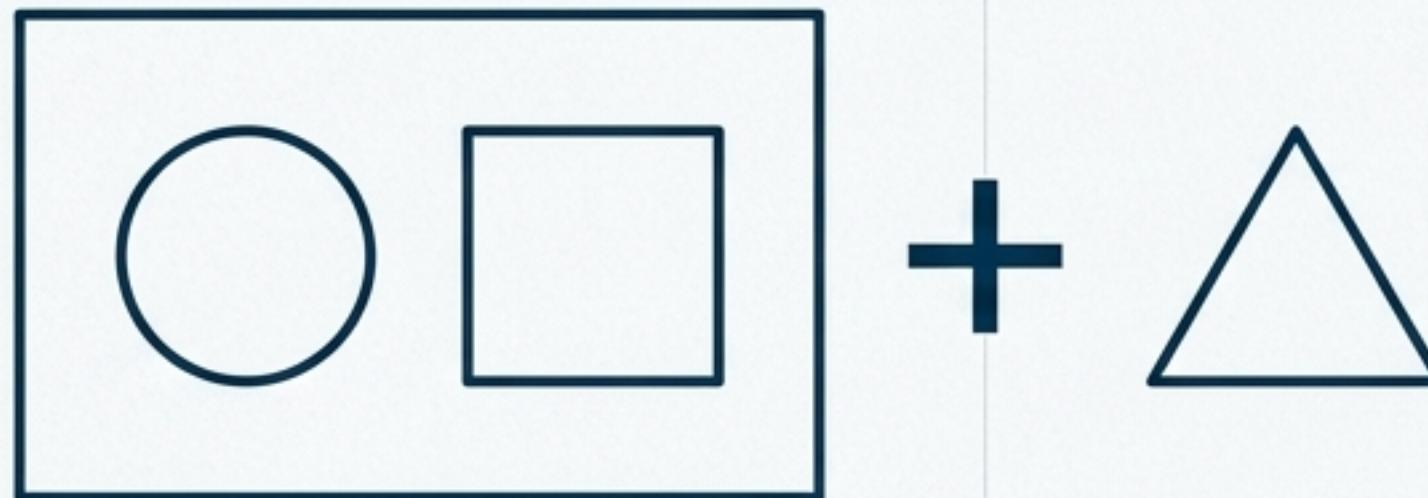
Your Mission

- .1 کلاس `Employee` را به یک کلاس `abstract` تبدیل کنید.
- .2 متدهای `calculatePay` را به یک متدهای `abstract` تبدیل کنید.
- .3 دو کلاس فرزند `calculatePay` که `SalariedPay` و `SalariedEmployee` بسازید را اشکل صحیح
- .4 دو کلاس فرزند `calculatePay` که `OvertimePay` و `HourlyEmployee` بسازید را به شکل صحیح و
مختص خودشان پیاده‌سازی کنند.





چالش دوم: گسترش نرم افزار نقاشی



The Context

ما یک سیستم عالی برای `Shape`‌ها داریم که `Shape` و `Circle` را پشتیبانی می‌کند.

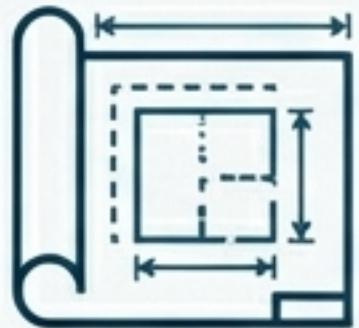
Your Mission

1. یک کلاس جدید به نام `Triangle` بسازید که از `Shape` ارث بری کند.
2. فیلدهای لازم (مانند `height` و `base`) را برای آن تعریف کنید.
3. متدهای `getPerimeter` و `getArea` را مطابق با قرارداد کلاس و با فرمول‌های صحیح برای مثلث، بازنویسی (Override) کنید.
4. در متod `main`، یک شیء از نوع `Triangle` به لیست `shapes` اضافه کنید و برنامه را اجرا کنید تا ثابت شود سیستم شما بدون هیچ تغییری در حلقه اصلی، شکل جدید را پشتیبانی می‌کند.





خلاصه قدرت‌های جدید شما



تعريف مفاهیم: اکنون می‌توانید کلاس‌های "مفهومی" تعریف کنید که قابل نمونه‌سازی نیستند (abstract class).



تحميل قرارداد: اکنون می‌توانید فرزندان را مجبور کنید تا رفتارهای خاصی را رفتارهای خاصی را پیاده‌سازی کنند (abstract method).



ترکیب کد مشترک و قرارداد: اکنون می‌توانید کد مشترک را به اشتراک بگذارید و پیناده بگذارید و همزمان پیاده‌سازی‌های سفارشی را الزامی کنید.



سلط بر چندریختی: شما اکنون پایه و اساس الگوهای طراحی قدرتمند و چندریختی (Polymorphism) را درک می‌کنید.





لودن لودن لودن

در بخش بعدی: قرارداد خالص



ما دیدیم که کلاس انتزاعی چگونه 'قرارداد' و 'کد مشترک' را ترکیب می‌کند.

اما اگر بخواهیم یک قرارداد ۱۰۰٪ خالص تعریف کنیم، بدون هیچ‌گونه پیاده‌سازی مشترک؟

با رابطها آشنا شوید – ابزاری برای تعریف تعهدات مطلق در دنیای جاوا.

