

بخش ۶-الف: سازندها و — آین تولد یک شیء

تهیه شده توسط: سید سجاد پیراهش

هرج و مرج قبل از تولد: چرا مقداردهی دستی خطرناک است؟

روش قدیمی و پرهرج و مرج

```
// The chaotic, manual way
Student s1 = new Student();
s1.name = "Ali";
s1.studentId = 987;
s1.major = "Computer Science"; ←
// What if we forgot to set the major? ⚠
```

تاکنون، ما اشیاء را اینگونه خلق می‌کردیم: ابتدا یک شیء "خام" می‌ساختیم و سپس فیلدهای آن را خط به خط پر می‌کردیم.
این روش سه مشکل اساسی دارد:



پرحرفی (Verbose): نیاز به نوشتندگی تکراری و طولانی.



مستعد خطا (Error-Prone): فراموش کردن مقداردهی یک فیلد بسیار آسان است.



وضعیت نامعتبر (Invalid State): برای لحظاتی، شیء ما در یک وضعیت ناقص و غیرقابل استفاده قرار دارد.

راه حل: سازنده (Constructor)، گواهی تولد شیء

سازنده یک بلوک کد ویژه است که تضمین می‌کند یک شیء در همان لحظه تولد، به یک وضعیت معتبر و کامل برسد. این آیین، مقداردهی اولیه (Initialization) را اجباری و خودکار می‌کند.



قوانین طلایی تعریف سازنده

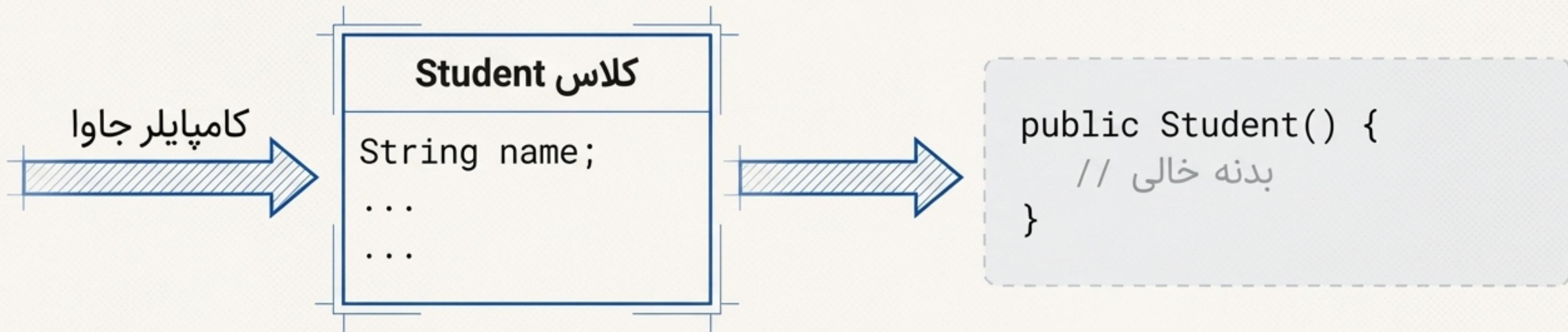
۱. نام سازنده **باید** دقیقاً با نام کلاس یکسان باشد.
۲. سازنده هیچ نوع خروجی (Return Type) ندارد، حتی `void`.



کالبدشکافی: تفاوت‌های کلیدی سازنده و متده

معیار	Method (متده)	Constructor (سازنده)
هدف	مقداردهی اولیه یک شئ	تعریف رفتار یک شئ
نام	باید دقیقاً با نام کلاس یکسان باشد	می‌تواند هر نام معتبری (camelCase) باشد
نوع خروجی	ندارد (هیچ، حتی void) 	دارد (باید مشخص شود، void هم مجاز است)
فراخوانی	ضمنی، هنگام استفاده از `new`	صريح، با استفاده از نام متده
وراثت	به ارث برده نمی‌شود	به ارث برده نمی‌شود (مگر private)

دستیار پنهان: سازنده پیشفرض (Default Constructor)

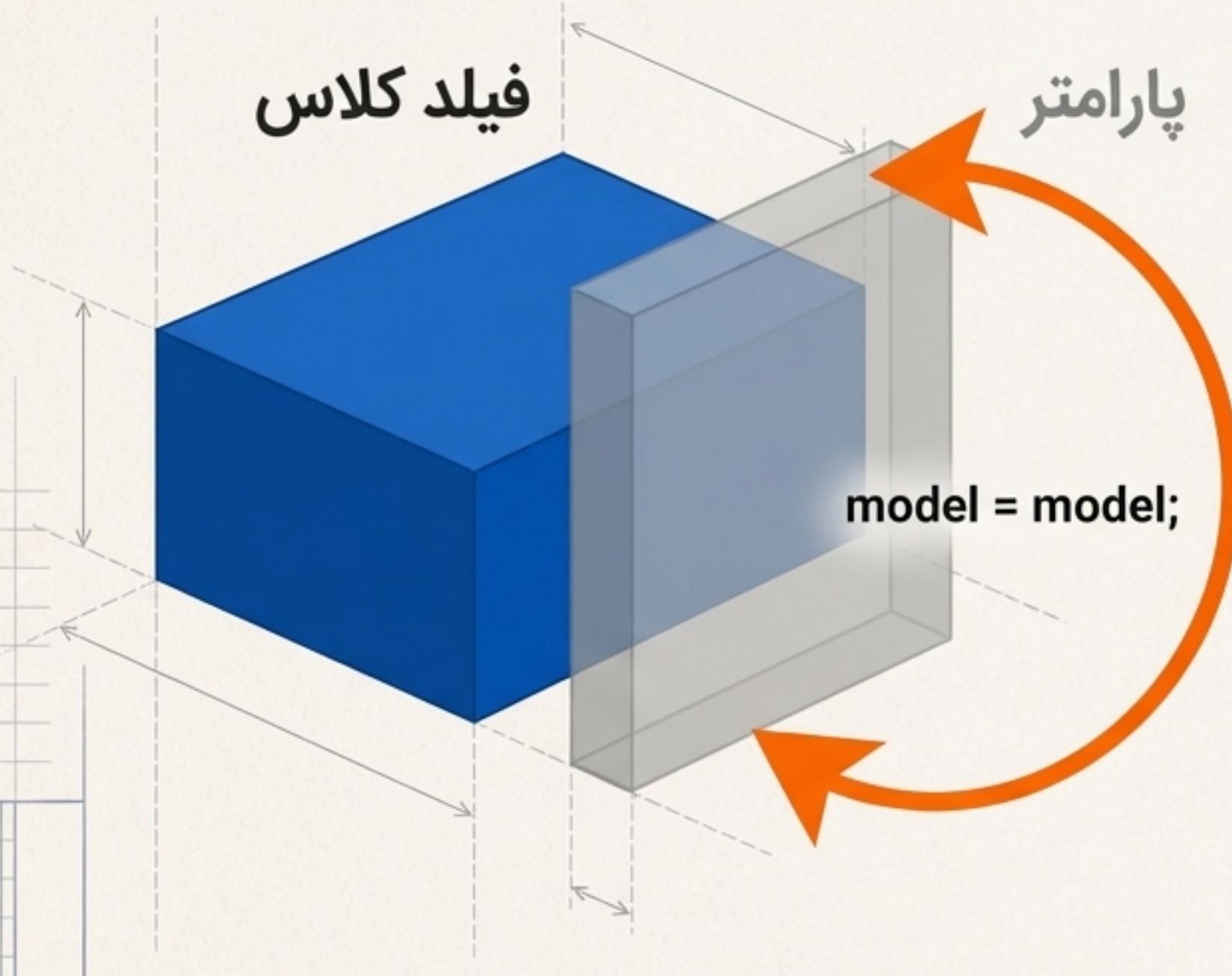


قانون مهم: اگر شما در کلاس خود هیچ سازنده‌ای ننویسید، کامپایلر جاوا به صورت خودکار یک سازنده پیشفرض (بدون پارامتر و با بدن خالی) برای شما ایجاد می‌کند. این همان چیزی است که به `()` new اجازه کار می‌دهد.



نقطه کور: لحظه‌ای که شما حتی یک سازنده در کلاس خود بنویسید (مثلاً یک سازنده با پارامتر)، کامپایلر دیگر سازنده پیشفرض را برای شما ایجاد نمی‌کند. این یک منبع شایع برای خطا است!

بحران هویت: وقتی نام پارامتر و فیلد یکسان است (Shadowing)



وقتی نام پارامتر یک سازنده با نام یک فیلد کلاس یکسان باشد، مشکلی به نام سایه‌افکنی (Shadowing) رخ می‌دهد. پارامتر محلی، فیلد کلاس را "پنهان" می‌کند و مقداردهی به درستی انجام نمی‌شود.

```
public class Car {  
    String model;
```

اشتباه محلی با داخلو، می شود:

```
public Car(String model) {
```

اشتباه! اینجا پارامتر به خودش تخصیص داده می‌شود.

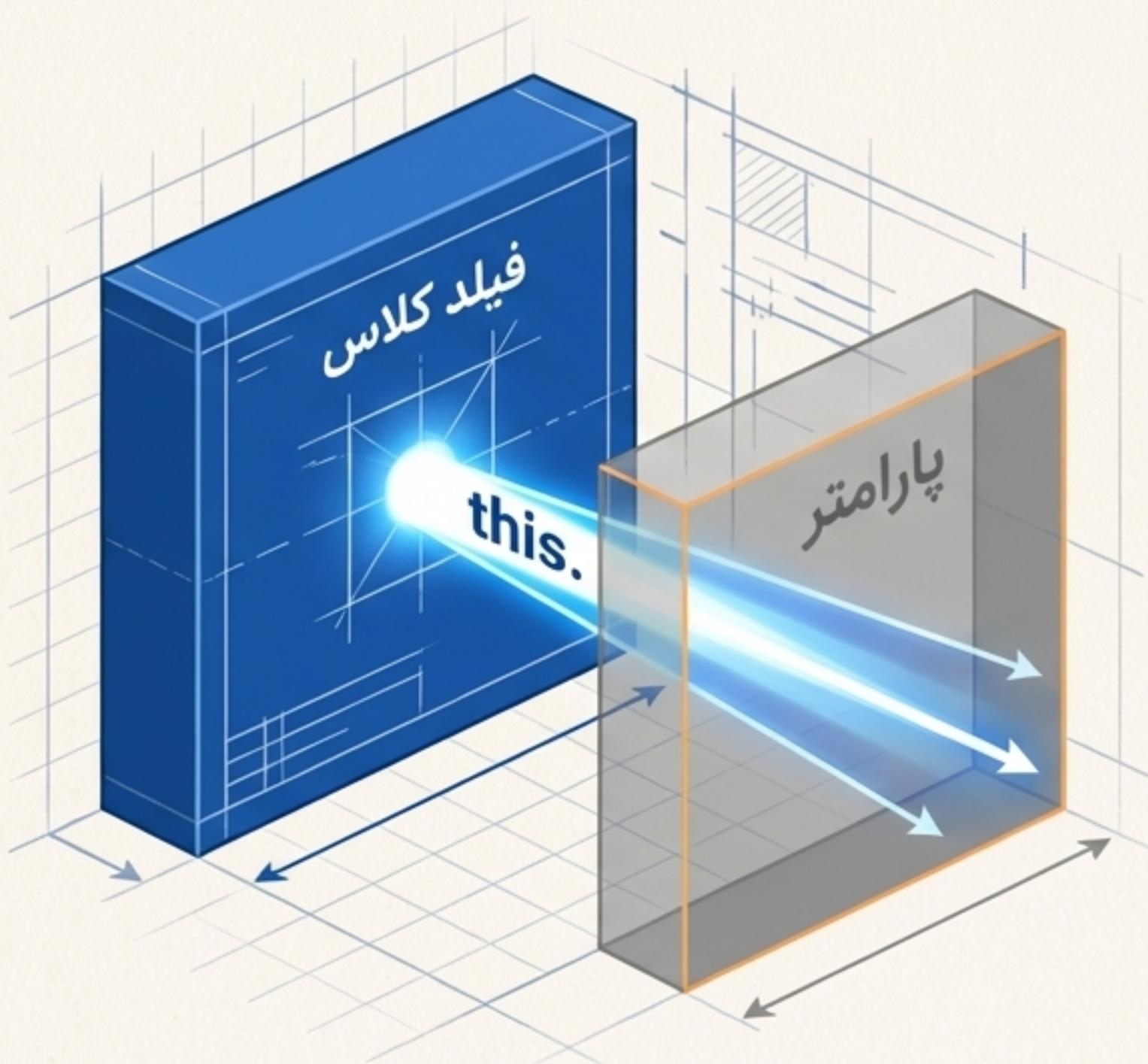
```
    model = model;
```

```
}
```

```
}
```

خروجی: فیلد model این ماشین null باقی می‌ماند! //

ابزار خودآگاهی: کلمه کلیدی `this`



کلمه کلیدی `this` یک ارجاع به شیء فعلی است. این ابزار به شیء اجازه می‌دهد تا به فیلد‌های خودش اشاره کند و مشکل سایه‌افکنی را حل کند.

- به فیلد `model` متعلق به این شیء اشاره دارد.
- به پارامتر `model` اشاره دارد.

```
public class Car {  
    String model;  
    public Car(String model) {  
        // مقداردهی می‌شود  
        this.model = model;  
    }  
}
```

صحیح! فیلد کلاس با پارامتر ورودی مقداردهی می‌شود!

چهار چهره قدرتمند `this`

کاربرد	توضیح	مثال
this.field	دسترسی به فیلد کلاس برای رفع ابهام (Shadowing)	this.name = name;
this.method()	فراخوانی متد دیگری از همان کلاس	this.displayInfo();
this()	فراخوانی سازنده دیگری از همان کلاس (Constructor Chaining)	this(name, 18);
return this;	برگرداندن شیء فعلی برای فعال کردن زنجیره متد (Method Chaining)	return this;

آیین کارایی: جلوگیری از تکرار کد با زنجیره سازنده (this())

همانند متدها، سازنده‌ها نیز می‌توانند سربارگذاری (Overload) شوند. برای جلوگیری از تکرار کد مقداردهی در سازنده‌های مختلف، می‌توانیم از داخل یک سازنده، سازنده اصلی‌تر را با () فراخوانی کنیم.

```
public Student(String name) {  
    this.name = name; // Redundant ⚡  
    this.age = 18;  
}
```

```
public Student(String name, int age) {  
    this.name = name; // Redundant ⚡  
    this.age = age;  
}
```

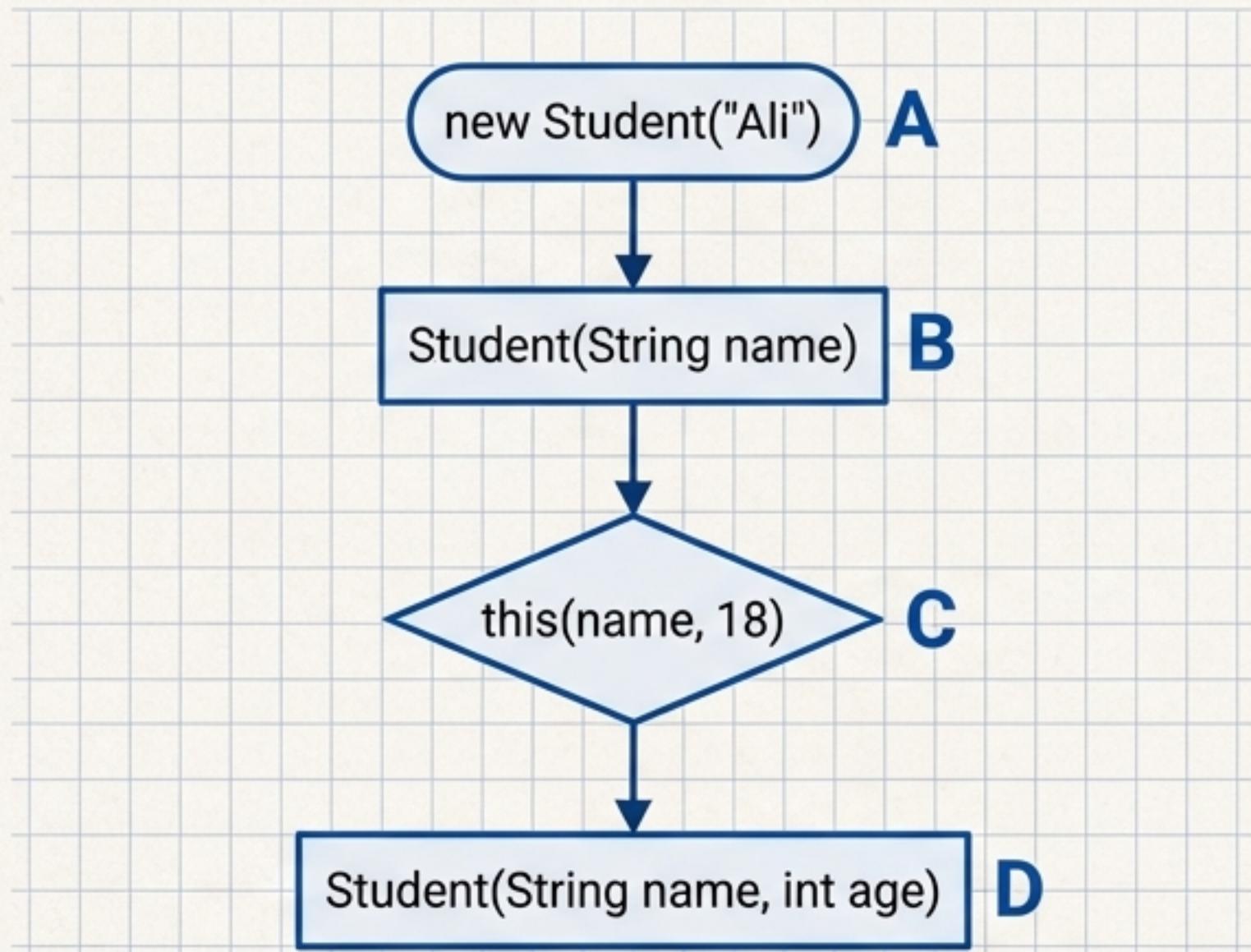
مشکل: تکرار کد

```
public class Student {  
    String name;  
    int age;  
  
    // The main, "master" constructor  
    public Student(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    // Overloaded constructor that calls the master  
    public Student(String name) {  
        // Instead of repeating code, we call the other constructor  
        this(name, 18); // Assume default age is 18  
    }  
}
```

راه حل: زنجیره سازنده با ()

راه حمل: زنجیره سازنده
the master constructor

تجسم زنجیره: جریان فراخوانی `this()`



قانون طلایی (`this(): فراخوانی `this)` باید اولین دستور در بدنه سازنده باشد.

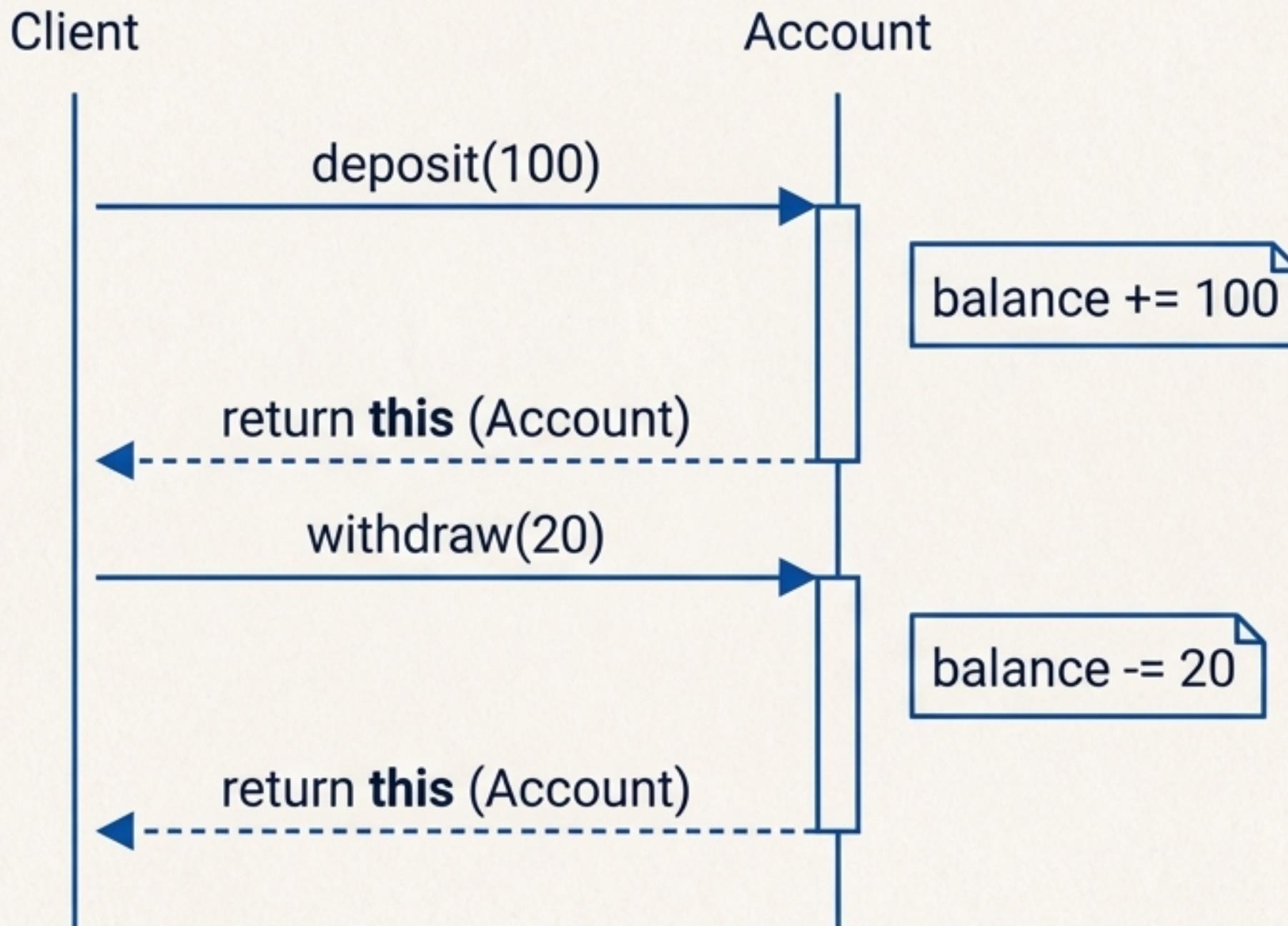
الگوی پیشرفته: خلق API‌های روان با زنجیره متدها (Method Chaining)

یک الگوی قدرتمند که در آن متدها به جای `void`، خود شیء (this) را برمی‌گردانند. این کار به ما اجازه می‌دهد تا می‌دهد تا فراخوانی متدها را مانند یک جمله روان و خوانا به هم زنجیر کنیم.

```
public class BankAccount {  
    private double balance;  
  
    public BankAccount deposit(double amount) {  
        this.balance += amount;  
        return this; // Return the current object  
    }  
  
    public BankAccount withdraw(double amount) {  
        this.balance -= amount;  
        return this; // Return the current object  
    }  
  
    // Fluent API Usage:  
    // account.deposit(100).withdraw(20).deposit(50);
```



تجسم جریان: آناتومی یک فراخوانی زنجیره‌ای



الگوی خاص: ساخت یک کپی بی نقص با سازنده کپی (Copy Constructor)



`title`
`author`



`title`
`author`

Perfect Copy

سازنده کپی، سازنده‌ای است که یک شیء دیگر از همان کلاس را به عنوان ورودی می‌گیرد و یک شیء جدید با مقادیر کاملاً یکسان ایجاد می‌کند. این روش برای " شبیه‌سازی " یا " کلون کردن " اشیاء بسیار مفید است.

```
public class Book {  
    String title;  
    String author;  
    // Normal constructor...  
    public Book(String title, String author) {  
        this.title = title;  
        this.author = author;  
    }  
  
    // The Copy Constructor  
    public Book(Book other) {  
        this.title = other.title;  
        this.author = other.author;  
    }  
}
```

کارگاه: اشکال‌یابی و دام‌های رایج



چالش اشکال‌یابی

سوال: اشکال کد زیر چیست و چرا فیلد `model` به درستی مقداردهی نمی‌شود؟

```
public class Car {  
    String model;  
    public Car(String model) {  
        model = model; // مشکل کجاست ?  
    }  
}
```



اشتباهات رایج که باید از آنها دوری کرد

1. فراموش کردن `this`: منجر به سایه‌افکنی (Shadowing) می‌شود.

2. فراخوانی `()` در جای اشتباه: باید اولین دستور در سازنده باشد.

3. تعریف سازنده با نوع خروجی: کامپایلر آن را یک متاد معمولی در نظر نمی‌گیرد، نه سازنده!

خلاصه آیین تولد شیء

- سازنده (Constructor) سازنده کد ویژه برای مقداردهی اولیه شیء؛ همنام کلاس و بدون نوع خروجی.
- سازنده پیشفرض سازنده پیشفرض: هدیه کامپایلر در صورت عدم وجود هرگونه سازنده دیگر.
- `this` ارجاعی به شیء فعلی برای حل ابهام و دسترسی به اعضای کلاس.
- `this()` ابزار زنجیره سازنده برای جلوگیری از تکرار کد (باید اولین دستور باشد).
- سربارگذاری سازنده ایجاد چندین راه برای ساخت یک شیء.
- Method Chaining کلید ساخت API‌های روان و زنجیره‌ای (`return this;`)

با تسلط بر این آیین، شما تضمین می‌کنید که هر شیء در برنامه شما زندگی خود را در یک وضعیت کامل و معتبر آغاز می‌کند.

→ در بخش بعدی: پکیج‌بندی حرفه‌ای و سطوح دسترسی جامع.