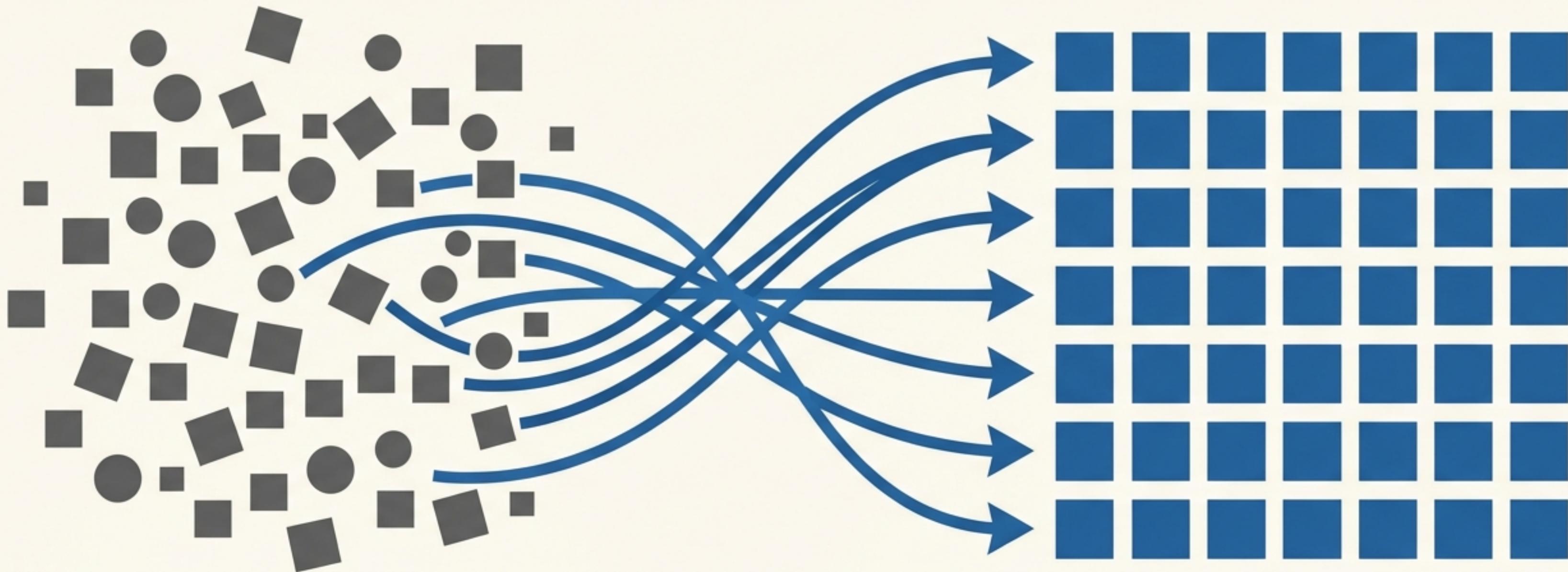


بخش ۴: آرایه‌ها – مدیریت مجموعه‌های داده

تهییه شده توسط: سید سجاد پیراھش



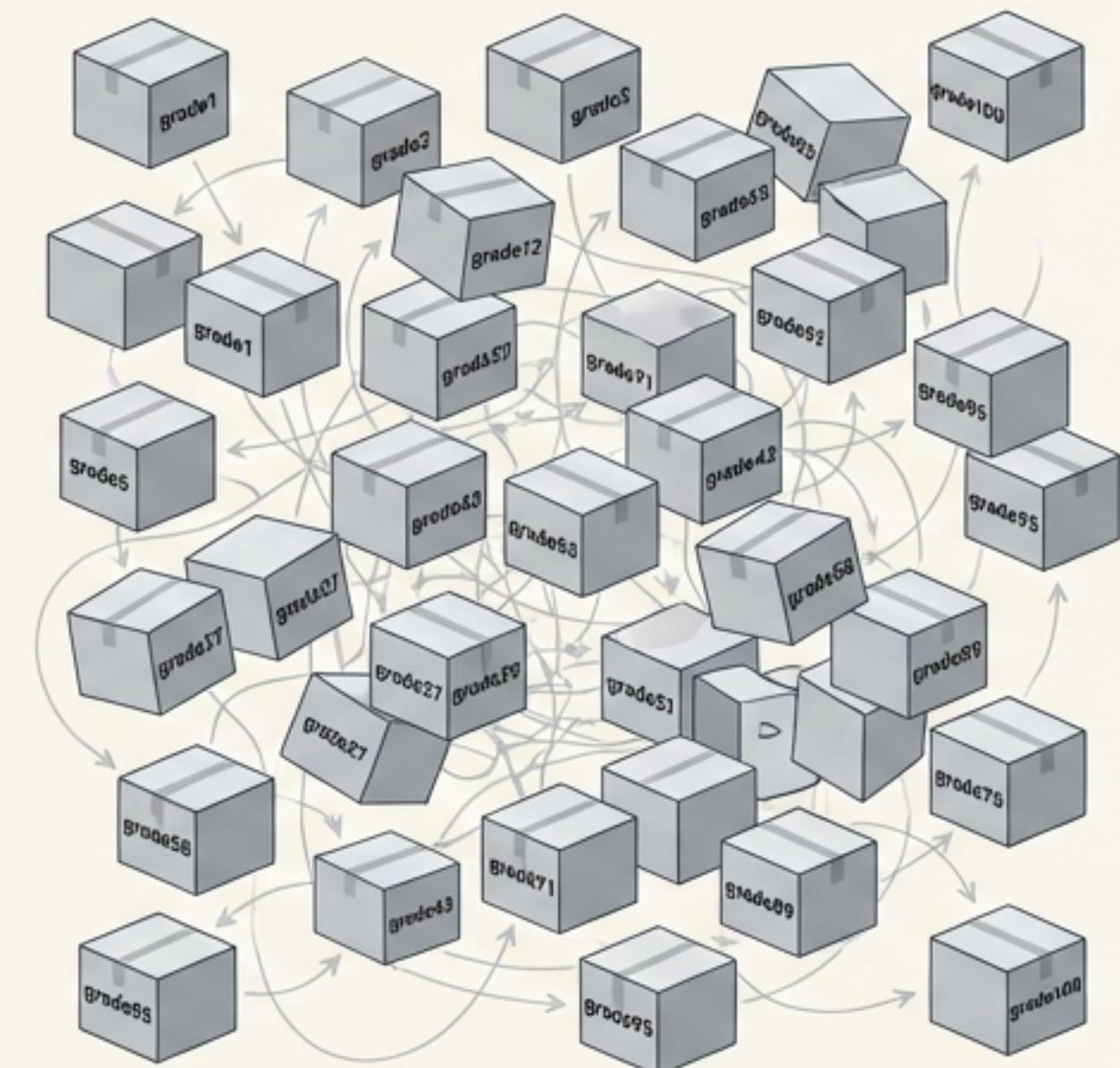
مشکلی که نمی‌دانستید دارید: کابوس ۱۰۰ متغیر

تا به حال، هر متغیر فقط یک مقدار را نگه می‌داشت. اما اگر بخواهیم نمرات ۱۰۰ دانشجو را ذخیره کنیم، آیا باید این کار را انجام دهیم؟

```
int grade1, grade2, grade3, ..., grade100;
```

نقاط ضعف این روش:

- **غیرقابل مدیریت:** پیدا کردن، بهروزرسانی یا حذف یک نمره خاص فاجعه است.
- **مستعد خطا:** احتمال اشتباه تایپی و منطقی بسیار بالاست.
- **پردازش غیرممکن:** چطور می‌توان روی این متغیرها حلقه زد تا میانگین را حساب کرد؟ نمی‌توان!

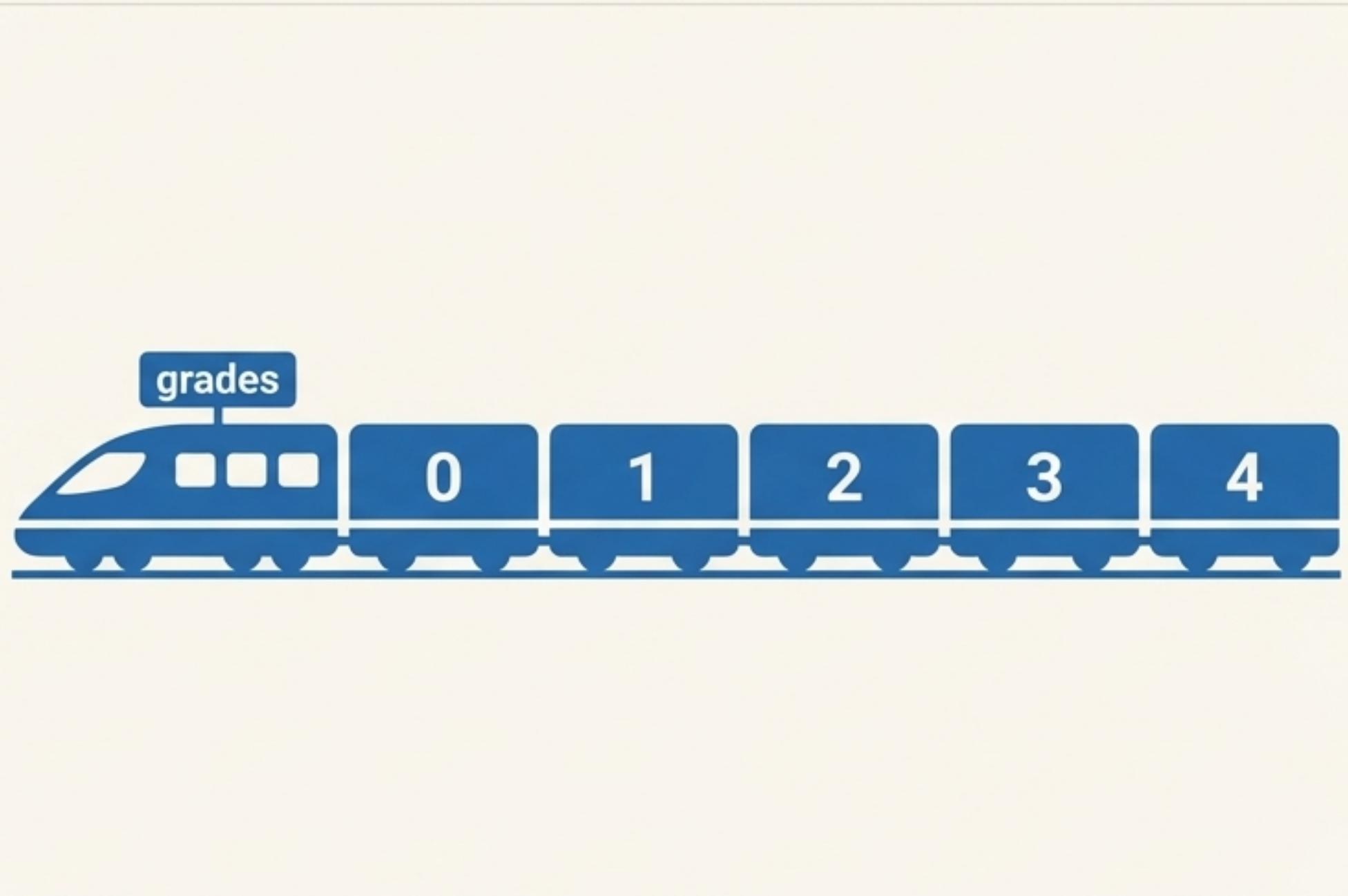


معرفی فهرمان: آرایه، سازماندهنده داده‌های شما

یک آرایه، یک مجموعه مرتب. از مقادیر است که همگی از یک نوع داده هستند و در خانه‌های حافظه پشت سر هم قرار می‌گیرند.

به زبان ساده، آرایه مانند یک قطار است:

- یک نام واحد (نام قطار) برای کل مجموعه.
- واگن‌های متصل و شماره‌دار (عناصر) که داده‌ها را حمل می‌کنند.
- تمام واگن‌ها یک نوع بار مشخص (نوع داده یکسان) را حمل می‌کنند.



آیین‌نامه رفتاری آرایه: سه قانون خدشه‌ناپذیر



(Fixed Size)

وقتی یک آرایه با اندازه‌ای مشخص ایجاد شد، اندازه آن **هرگز*** قابل تغییر نیست. این مهمترین ویژگی و محدودیت آرایه‌هاست.



(Homogeneous)

یک آرایه `int` فقط می‌تواند `int` نگه دارد. نمی‌توانید یک `String` را در قطار اعداد صحیح صحیح قرار دهید.

[0] [1] [2] [3]



(Indexed Access)

هر واگن (عنصر) یک شماره منحصر به فرد از ۰ تا ۱ - `length` دارد. این آدرس دقیق آن برای دسترسی سریع است.

ویژگی	متغیرهای مجرزا	آرایه
تعداد نام	هر متغیر یک نام	یک نام برای همه
مدیریت	بسیار سخت	ساده
حلقه‌زدن	غیرممکن	آسان
مثال	score1, score2, ...	scores[0], scores[1], ...

منبع قدرت: نگاهی به دنیای مخفی حافظه

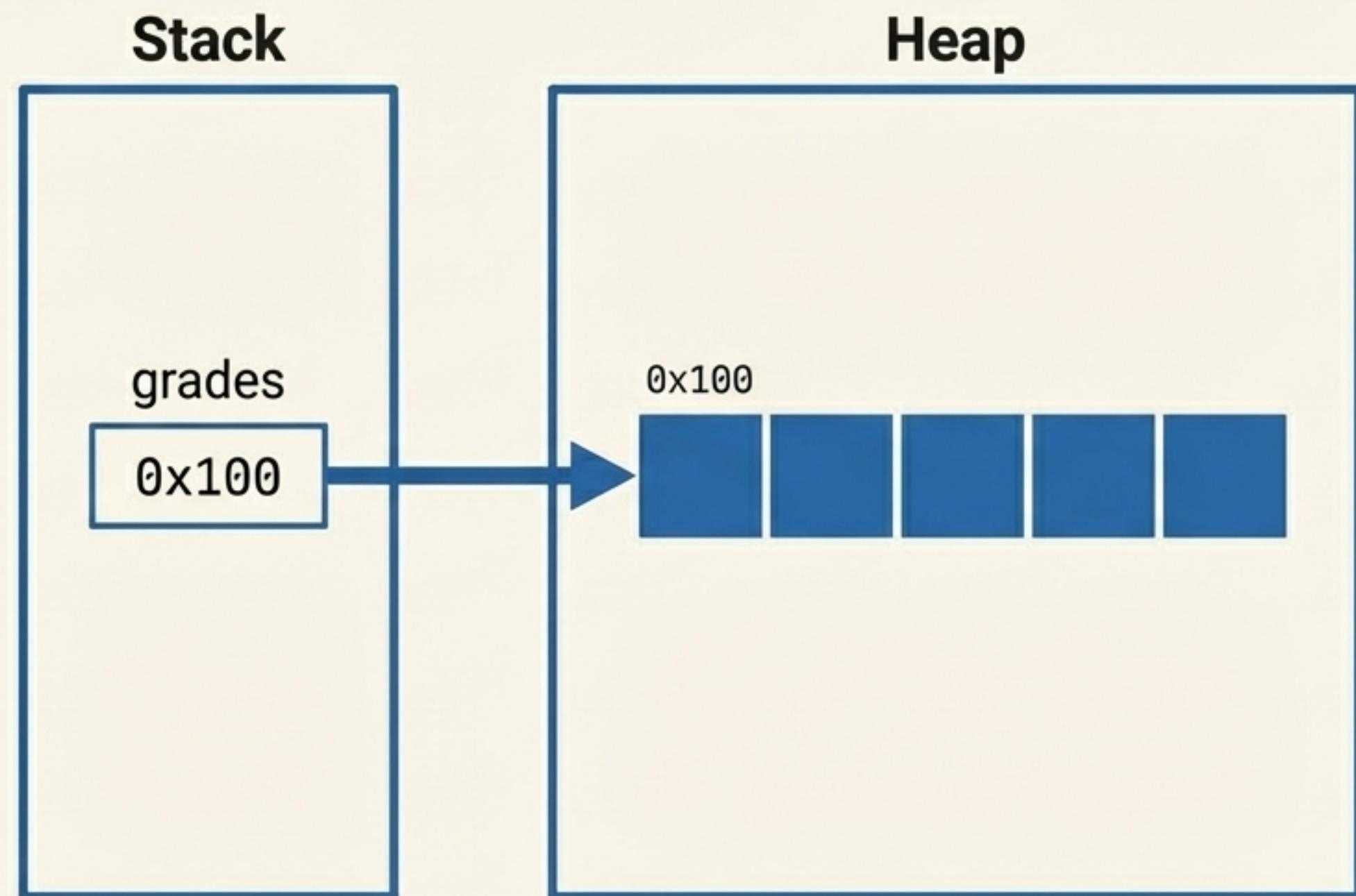
مهمترین نکته فنی: آرایه‌ها در جاوا شیء (Object) هستند.

این یعنی مدیریت حافظه دو مرحله‌ای است:

1. **متغیر ارجاعی (Reference Variable)**: نام آرایه شما (مثلاً `grades` مانند یک آدرس روی یک کاغذ است. این آدرس در حافظه Stack زندگی می‌کند.

2. **شیء آرایه (Array Object)**: خود قطار داده‌ها (ساختار اصلی) در یک فضای بزرگ به نام حافظه Heap ساخته می‌شود.

متغیر در Stack فقط آدرس شروع آرایه در Heap را نگه می‌دارد.



نقشه فنی یک آرایه در حافظه

وقتی این کد اجرا می‌شود:

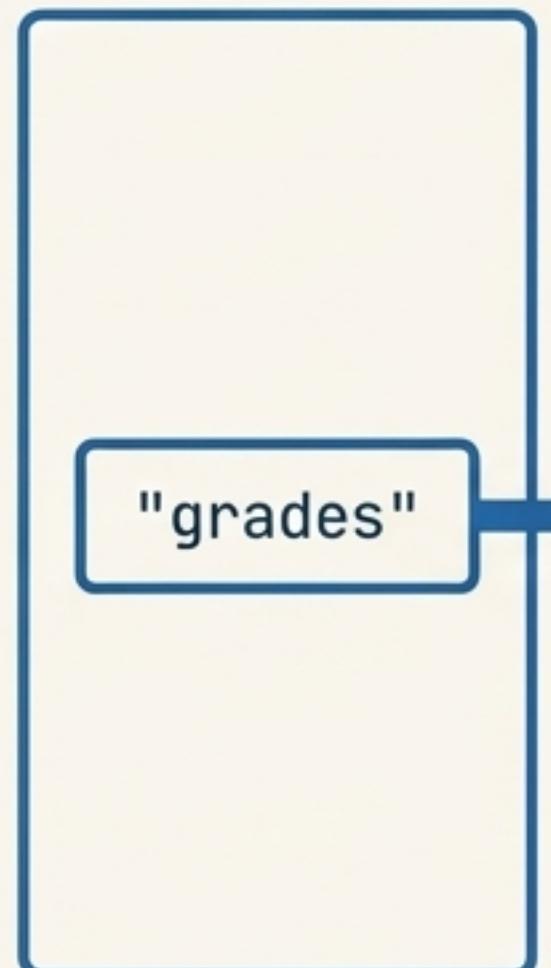
```
int[] grades = new int[4];
```

یک آرایه در Heap ایجاد می‌شود که می‌تواند ۴ مقدار `int` را نگه دارد.

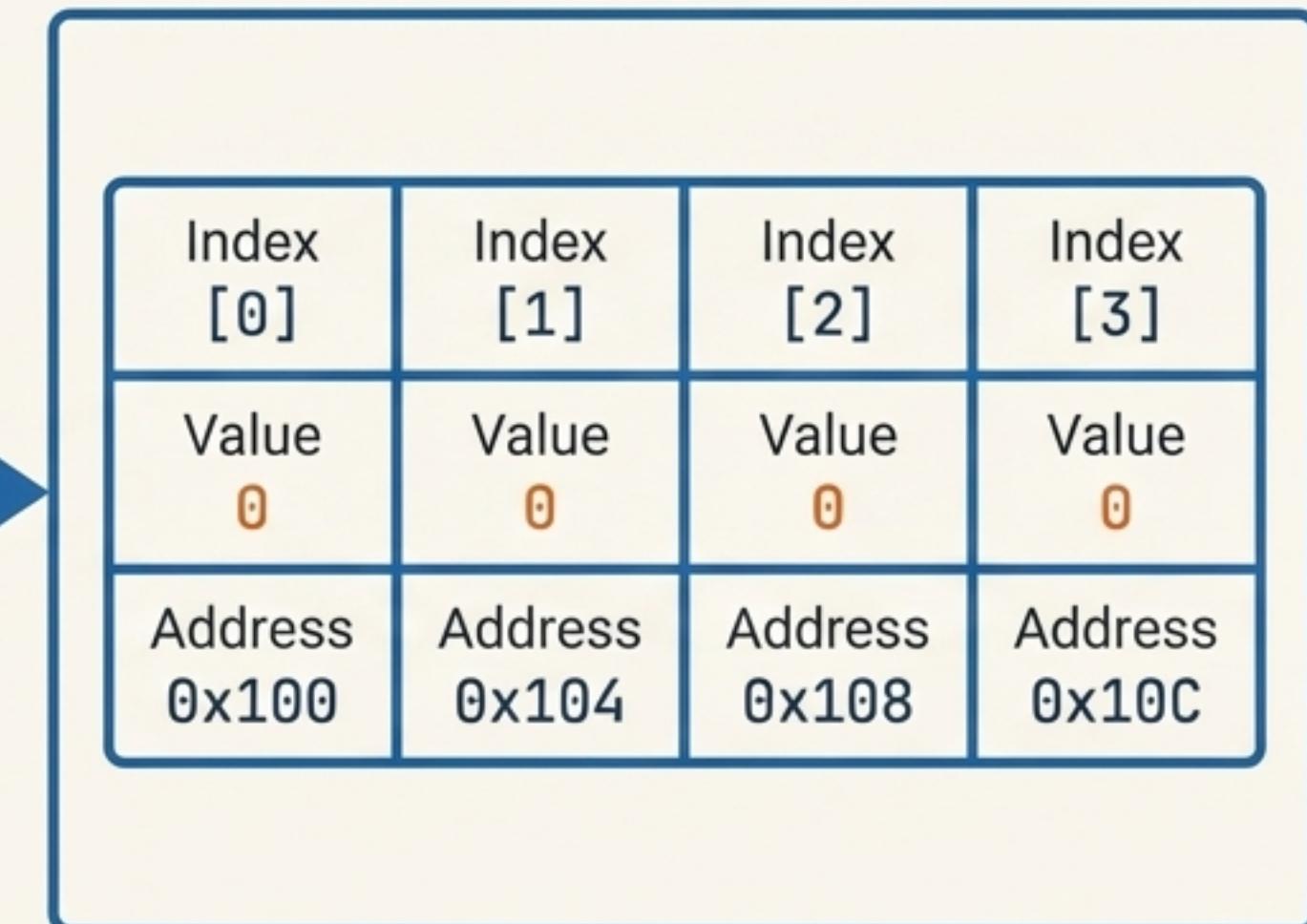
جاوا به طور خودکار تمام خانه‌ها را با مقدار پیش‌فرض آن نوع داده پر می‌کند.

نوع آرایه	مقدار پیش‌فرض
int[], long[], byte[]	0
float[], double[]	0.0
boolean[]	false
(String[]) آرایه اشیاء	null

****Stack****



****Heap****



جان بخشیدن به آرایه‌ها: از ایده تا اجرا

فرآیند کار با آرایه‌ها سه مرحله کلیدی دارد:

1

اعلان (Declaration): به کامپایلر می‌گوییم چه چیزی در راه است.

```
// یک رفرنس null در Stack ایجاد می‌شود  
String[] names;
```

2

ایجاد (Instantiation): با `new`، شیء آرایه را در Heap می‌سازیم.

```
// آرایه‌ای با ۳ خانه در Heap ساخته شد  
names = new String[3];
```

3

مقداردهی (Initialization): خانه‌ها را پر می‌کنیم.

```
names[0] = "سارا";  
names[1] = "علی";  
names[2] = "مریم";
```

Code Sandbox

```
double[] scores = { 98.5, 89.0, 76.75, 100.0 };
```

میانبر قدرتمند: Array Literal

می‌توان هر سه مرحله را در یک خط انجام داد:

اتحاد کامل: چرا آرایه‌ها و حلقه‌ها هم ساخته شده‌اند؟

قدرت واقعی آرایه زمانی آزاد می‌شود که آن را با بهترین دوستش، یعنی حلقه، همراه کنید.

اصل DRY (Don't Repeat DRY): به جای نوشتن کدهای تکراری برای هر عنصر، یک بار برای همیشه منطق را در یک حلقه می‌نویسیم.

الف) حلقه `for` کلاسیک (زمانی که به شاخص نیاز دارد)

```
// چاپ نمرات به همراه شماره دانشجو
for (int i = 0; i < scores.length; i++) {
    System.out.println("نمره نفر" + (i+1) + ":" + scores[i]);
}
```

تکرار نکنید!

```
// grades[0] = ...
// grades[1] = ...; <-- تکرار نکنید!
// grades[2] = ...;
```

ب) حلقه `for-each` پیشرفته (زمانی که فقط به مقدار نیاز دارد)

```
محاسبه مجموع نمرات //
double sum = 0;
for (double score : scores) {
    sum += score;
}
```

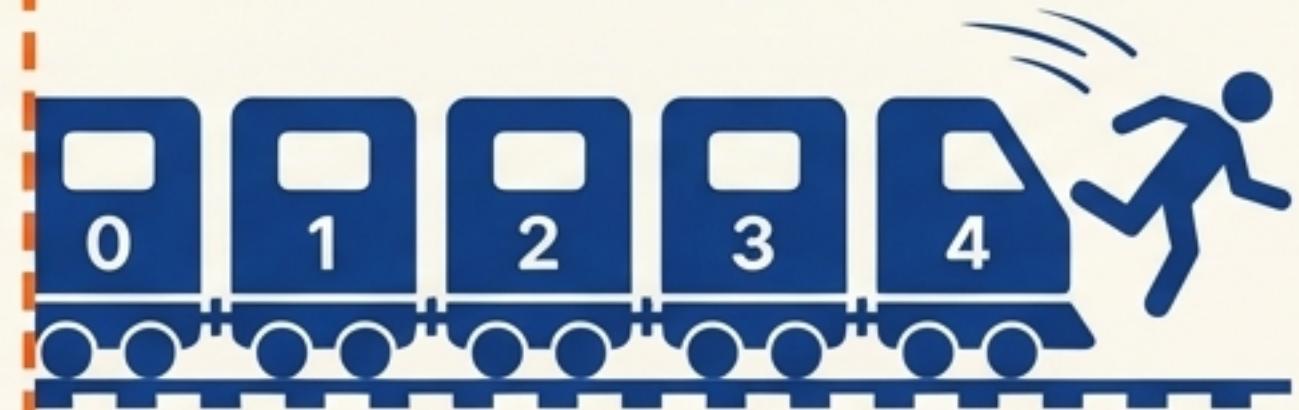
for-each	for کلاسیک	ویژگی
✗	✓	دسترسی به index
⚠ (فقط محتوای اشیاء)	✓	تغییر عناصر
عالی	متوسط	خوانایی
فقط خواندن مقادیر	نیاز به index، تغییر مقدار	کاربرد

منطقه خطر! رایج‌ترین خطای آرایه‌ها

مهمکننده‌ترین خطای زمان اجرا: `ArrayIndexOutOfBoundsException`

دلیل: شاخص‌های یک آرایه با اندازه N از ۰ تا $N-1$ هستند. تلاش برای دسترسی به شاخصی خارج از این محدوده (مانند N یا -1) باعث این خطا می‌شود.

استعاره: شما یک قطار با ۵ واگن دارید (شاخص‌های ۰ تا ۴). اگر تلاش کنید به واگن شماره ۵ دسترسی پیدا کنید، از انتهای قطار به بیرون پرتاپ می‌شوید!



مثال کد خطا

```
int[] numbers = new int[5]; // شاخصها: 0, 1, 2, 3, 4  
System.out.println(numbers[5]); // BOOM! Exception
```

// اشتباه رایج در حلقه

```
for (int i = 0; i <= numbers.length; i++) { // < numbers.length  
    // ...  
}
```

سطح بالاتر: آرایه‌های چندبعدی

جاوا مستقیماً آرایه چندبعدی ندارد. در عوض، یک مفهوم هوشمند -نانه‌تر را پیاده‌سازی می‌کند: **آرایه‌ای از آرایه‌ها** (Array of Arrays).

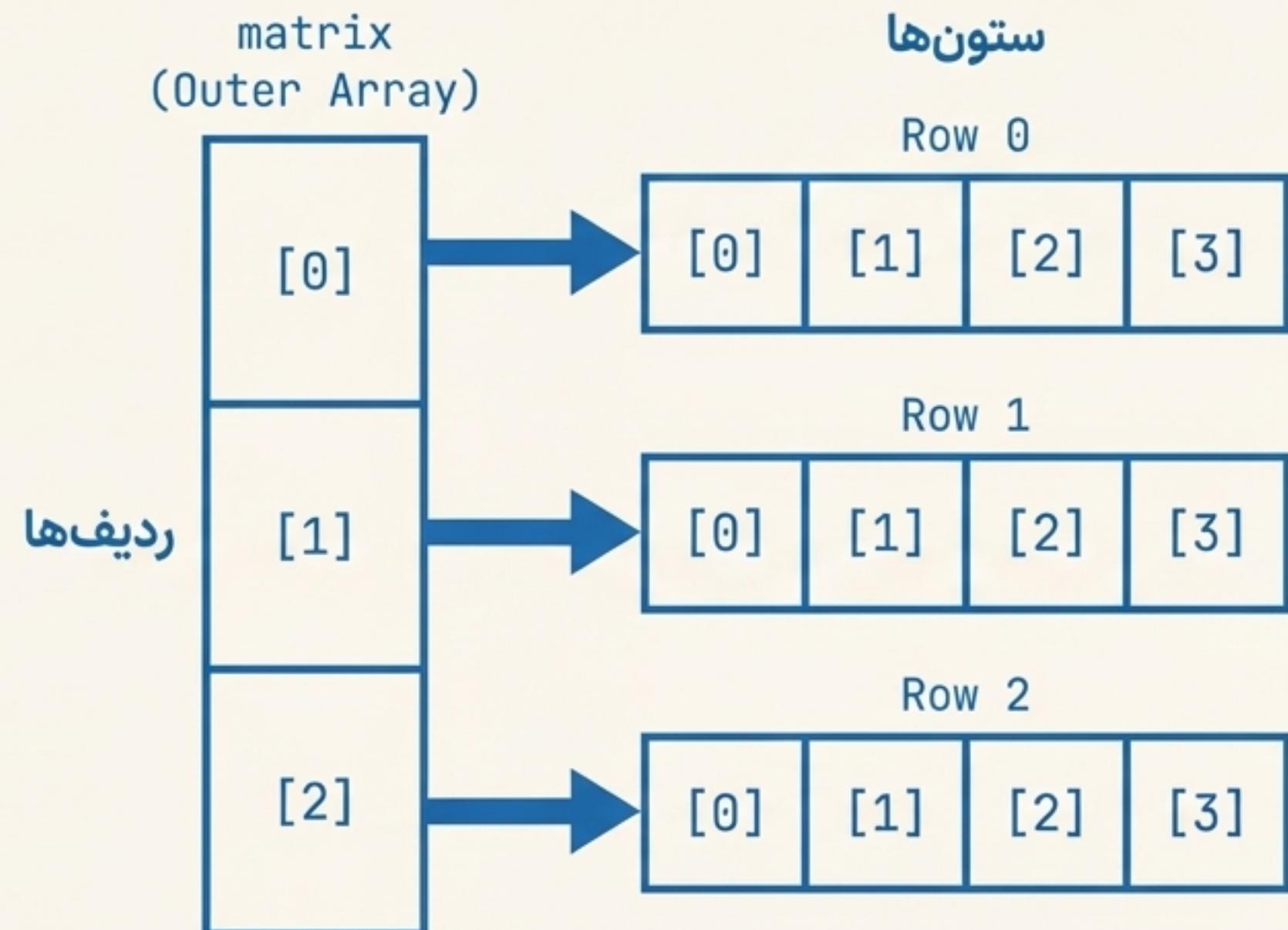
یک آرایه دو بعدی را مانند یک ساختمان یا سالن سینما تصور کنید:

- * **آرایه بیرونی**: لیستی از تمام طبقات (ردیف‌ها).
- * **هر عنصر آرایه بیرونی**: یک آرایه دیگر است که اتاق‌های آن طبقه (ستون‌ها) را مشخص می‌کند.

مثال تعریف:

```
// یک ماتریس با ۳ ردیف و ۴ ستون  
int[][] matrix = new int[3][4];
```

این کد یک آرایه به نام `matrix` با ۳ عنصر ایجاد می‌کند. هر عنصر، یک ارجاع به یک آرایه `int` با اندازه ۴ است.



آسمان خراش در عمل: منطق ماتریس

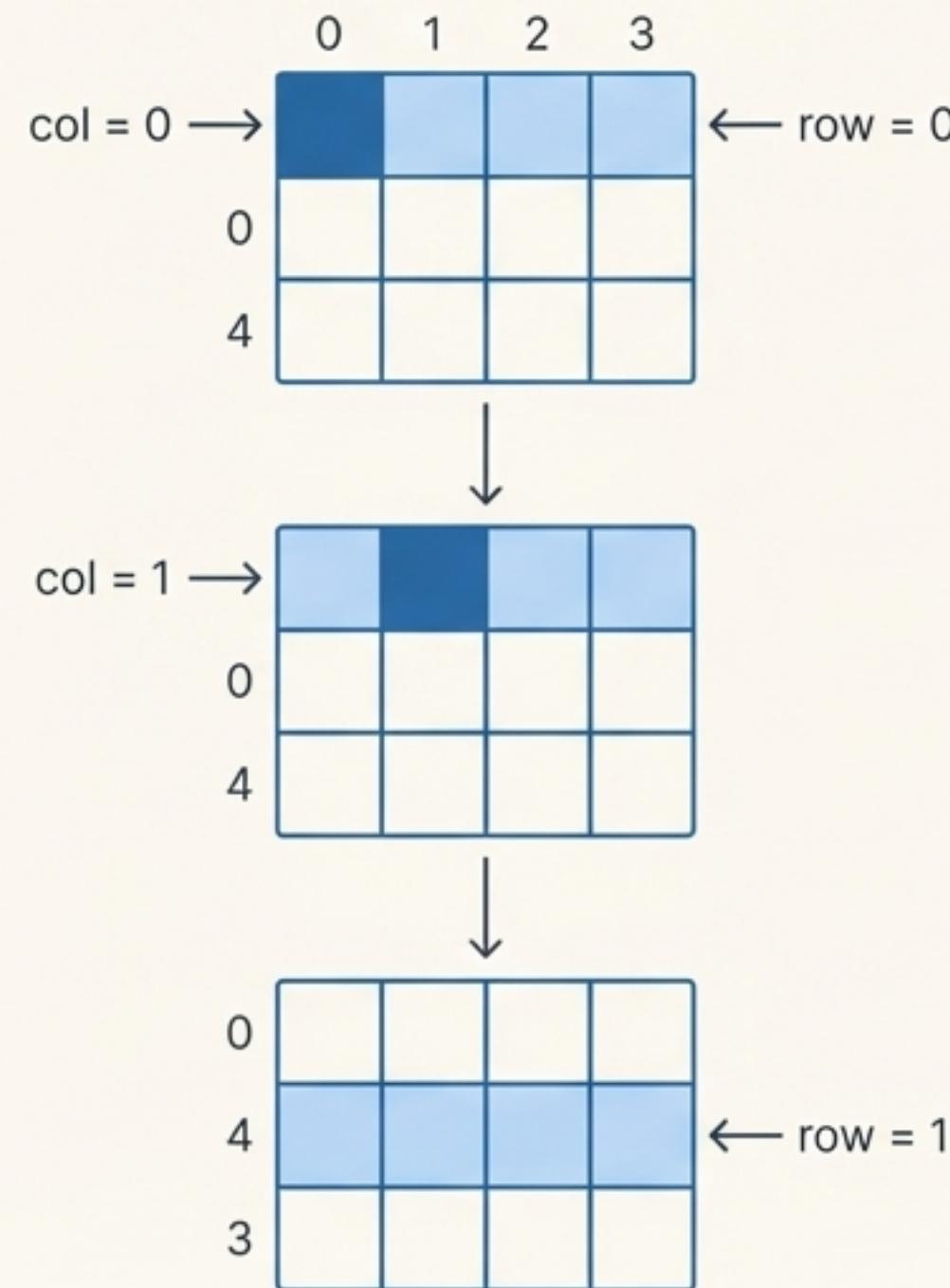
برای کار با هر عنصر در یک آرایه دو بعدی، به حلقه‌های تودرتو (Nested Loops) نیاز داریم. حلقه بیرونی ردیف‌ها را پیمایش می‌کند و حلقه درونی، ستون‌های هر ردیف را.

مقداردهی و چاپ یک ماتریس**

```
int[][] matrix = new int[3][4];
int counter = 1;
// پیمایش برای مقداردهی
for (int row = 0; row < matrix.length; row++) {
    for (int col = 0; col < matrix[row].length; col++) -
        matrix[row][col] = counter++;
}
// پیمایش برای چاپ //
for (int[] row : matrix) {
    for (int cell : row) {
        System.out.print(cell + "\t");
    }
    System.out.println();
}
```

خروجی

1	2	3	4
5	6	7	8
9	10	11	12



مرز نهایی: آرایه‌ای از اشیاء

وقتی آرایه‌ای از یک نوع کلاس (مانند Student یا Seat) می‌سازید، یک نکته بسیار حیاتی وجود دارد.

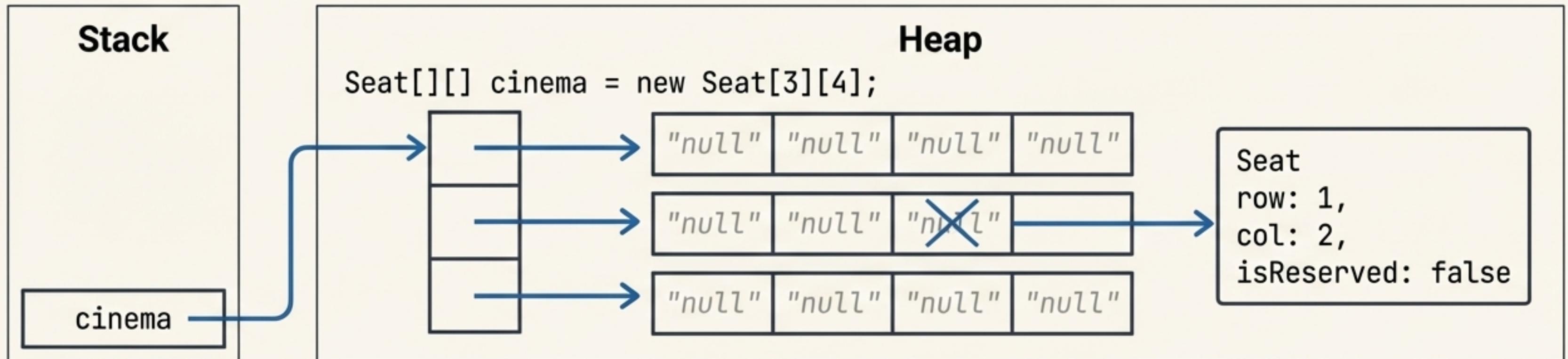
```
Student[] students = new Student[3];
```

۳ شیء Student ایجاد نمی‌کند!*** بلکه فقط یک آرایه (یک قفسه) در Heap می‌سازد که قادر است **۳* ارجاع (reference)** بلکه فقط یک آرایه (یک قفسه) در Heap می‌رَأَ به اشیاء Student را نگه دارد. در ابتدا، تمام این ۳ خانه null هستند.

1. ساختن آرایه (قفسه):

```
Student[] students = new Student[3];
```
2. ساختن هر شیء و قرار دادن آن در آرایه:

```
students[0] = new Student("آرش", 3.8);  
students[1] = new Student("پریسا", 3.9);  
students[2] = new Student("بهنام", 3.4);
```



میدان اثبات: مأموریت شما، اگر پذیرید...



میانگین نمرات

برنامه‌ای بنویسید که میانگین نمرات آرایه زیر را محاسبه و چاپ کند.

```
double[] grades = { 88.5,  
92.0, 75.5, 100.0, 81.0  
};
```



یافتن بزرگترین عدد

برنامه‌ای بنویسید که بزرگترین عدد در آرایه زیر را پیدا و چاپ کند.

```
int[] numbers = { 15, 4,  
29, 101, 8, 42 };
```



معکوس کردن آرایه

برنامه‌ای بنویسید که عناصر آرایه زیر را معکوس کرده و آرایه جدید را چاپ کند.

```
String[] words = {  
"Java", "is", "fun" };  
  
// Expected: {"fun", "is",  
"Java"}
```

غول آخر: چالش‌های ماتریس و اشیاء

$$\begin{bmatrix} 1 & 2 & 9 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 9 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

جمع دو ماتریس

برنامه‌ای بنویسید که دو ماتریس زیر را با هم جمع کرده و ماتریس حاصل را چاپ کند.

```
int[][] a = { {1, 2}, {3, 4} };  
int[][] b = { {5, 6}, {7, 8} };
```



آرایه دانشجویان

با استفاده از کلاس `Student` (که دارای فیلد‌های `name` و `gpa` است)، یک آرایه برای ۳ دانشجو بسازید. آنها را مقداردهی کرده و سپس نام دانشجویانی که معدل (gpa) آنها بالاتر از 3.5 است را چاپ کنید.

شما بر داده‌ها مسلط شدید: قدرت‌های جدید شما

خلاصه آنچه در این سفر آموختید و به جعبه ابزار خود اضافه کردید:

✓ آرایه: مجموعه مرتب و سازمان‌یافته از یک نوع داده.

✓ اندازه ثابت: پس از ایجاد، غیرقابل تغییر.

✓ شیء در Heap: آرایه‌ها اشیاء هستند و در Heap زندگی می‌کنند.

✓ شاخص 0 تا 1: آدرس دقیق هر عنصر.

✓ length: یک فیلد (بدون پرانتز) برای گرفتن اندازه.

✓ for-each در برابر for پیمایش با شاخص یا فقط با مقدار.

✓ آرایه اشیاء: ابتدا null، سپس باید هر شیء را ساخت و جایگزین کرد.

✓ چندبعدی: آرایه‌ای قدرتمند از آرایه‌ها.

نگاه به آینده

در بخش بعدی: متدها (Methods) – به اشیاء خود قدرت رفتار و عمل می‌دهیم.