

به نام خدا



برنامه‌سازی پیشرفته

دانشگاه شهید بهشتی · دانشکده مهندسی کامپیوتر

دکتر مجتبی وحیدی اصل

آرایه ها- قسمت دوم

نیما سلطانی

فهرست مطالب

1. آرایه های دو بعدی

2. آرایه های نامنظم

3. پردازش آرایه های چند بعدی

`varargs` .4

5. کلاس `Arrays`

انگیزه

تصور کنید وارد یک سینما شده‌اید و می‌خواهید بدانید کدام صندلی‌ها خالی هستند. اگر فقط یک ردیف صندلی داشتیم، آرایه‌ی تک‌بعدی کافی بود. اما وقتی سالن چندین ردیف و ستون دارد، به یک آرایه دو‌بعدی نیاز داریم.

جدول صندلی‌های سینما

۵	۴	۳	۲	۱	
■	■	■	■	■	ردیف A
■	■	■	■	■	ردیف B
■	■	■	■	■	ردیف C

در این جدول، ■ یعنی صندلی خالی و ■ یعنی صندلی رزرو شده. با یک آرایه دو‌بعدی می‌توانیم وضعیت کل سالن را به‌سادگی مدیریت کنیم.

اعلان / ایجاد آرایه‌های دوبعدی

آرایه‌های دو بعدی برای ذخیره سازی داده‌ها به صورت جدول یا ماتریس استفاده می‌شوند. در ادامه روش‌های مختلف اعلان و ایجاد آن‌ها را بررسی می‌کنیم.

۱. **اعلان آرایه دو بعدی:** در این روش فقط نوع و نام آرایه مشخص می‌شود، بدون تخصیص حافظه.

```
dataType[][] array_name;
```

۲. **ایجاد آرایه در heap و انتساب به متغیر:** در این مرحله حافظه اختصاص داده می‌شود و آرایه آماده استفاده است.

```
array_name = new dataType[10][10];
```

۳. ترکیب اعلان و ایجاد: این روش رایج‌ترین و توصیه‌شده‌ترین روش در جاوا است.

```
dataType[][] array_name = new dataType[10][10];
```

۴. روش جایگزین (سبک قدیمی‌تر): در این روش جایگاه براکت‌ها متفاوت است.

```
dataType array_name[][] = new dataType[10][10];
```

در جاوا، آرایه‌های دو بعدی در واقع آرایه‌ای از آرایه‌ها هستند. یعنی هر سطر یک آرایه مستقل است. این ساختار امکان ایجاد آرایه‌های نامتقارن (jagged arrays) را نیز فراهم می‌کند.

اعلان و مقداردهی آرایه‌های دوبعدی

۱. اعلان و ایجاد آرایه: دو سبک نحوی برای ایجاد آرایه وجود دارد:

```
int[][] matrix = new int[10][10];  
// or  
int matrix[][] = new int[10][10];
```

۲. مقداردهی دستی: می‌توان به صورت مستقیم به عناصر آرایه مقدار داد:

```
matrix[0][0] = 3;
```

این دستور مقدار ۳ را در خانه‌ی سطر صفرم و ستون صفرم قرار می‌دهد.

۳. مقداردهی با حلقه و مقادیر تصادفی: برای پر کردن آرایه با مقادیر تصادفی از حلقه‌های تو در تو استفاده می‌کنیم:

```
for (int i = 0; i < matrix.length; i++)
    for (int j = 0; j < matrix[i].length; j++)
        matrix[i][j] = (int)(Math.random() * 1000);
```

تابع `(Math.random())` عددی بین ۰ تا ۱ تولید می‌کند که با ضرب در ۱۰۰۰ و تبدیل به عدد صحیح، مقدار نهایی را می‌سازد.

اعلان آرایه‌های اعشاری: می‌توان آرایه‌هایی با نوع داده‌ی `double` نیز تعریف کرد:

```
double[][] x;
```

نکته مهم: هنگام استفاده از حلقه‌ها، از `matrix[i].length` برای تعداد سطرها و `matrix.length` برای تعداد ستونها استفاده کنید تا از خطای `ArrayIndexOutOfBoundsException` جلوگیری شود.

نمایی از آرایه دو بعدی

	0	1	2	3	4
0					
1					
2					
3					
4					

```
matrix = new int[5][5];
```

	0	1	2	3	4
0					
1					
2			7		
3					
4					

```
matrix[2][1] = 7;
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

```
int[][] array = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};
```

```
matrix = new int[5][5];
matrix[2][1] = 7;
```

```
int[][] array = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```

```
matrix.length      // 5
matrix[0].length   // 5
array.length       // 4
array[0].length    // 3
```

اعلان، ایجاد و مقداردهی آرایه‌ها با نمادهای میانبر

می‌توان آرایه‌های دو بعدی را به صورت هم‌زمان اعلان، ایجاد و مقداردهی کرد. این روش با استفاده از نماد `{ }` انجام می‌شود و خوانایی کد را افزایش می‌دهد.

مقداردهی اولیه با نماد خاص:

```
int[][] array = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};
```

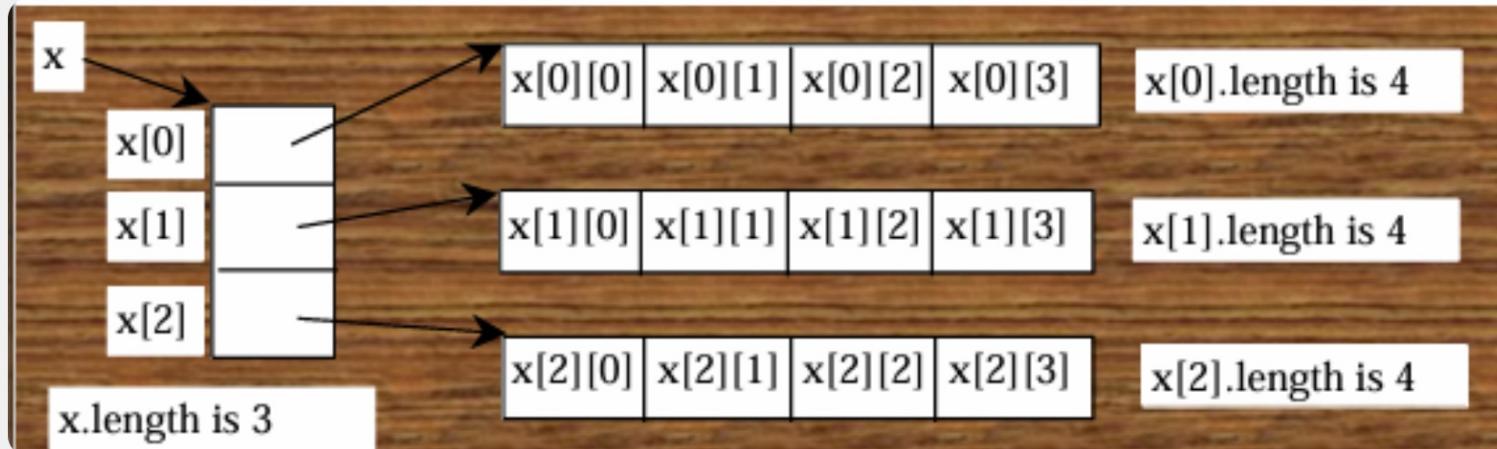
این روش هم‌زمان آرایه را ایجاد و مقداردهی می‌کند. طول سطرها و ستون‌ها به صورت خودکار تعیین می‌شود.

معادل با مقداردهی دستی:

```
int[][] array = new int[4][3];  
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;  
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;  
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;  
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

در این روش، ابتدا آرایه با ابعاد مشخص ایجاد می‌شود و سپس هر عنصر به صورت جداگانه مقداردهی می‌گردد.

طول آرایه‌های دو بعدی و دسترسی به زیرآرایه‌ها



۱. آرایه با ابعاد ثابت: در این حالت، تعداد سطرها و ستونها از ابتدا مشخص است.

```
int[][] x = new int[3][4];
x.length          // 3
x[0].length       // 4
x[1].length       // 4
x[2].length       // 4
```

۲. آرایه با مقداردهنده: در این روش، مقادیر اولیه هم زمان با ایجاد آرایه مشخص می‌شوند.

```
int[][] array = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};  
array.length      // 4  
array[0].length  // 3  
array[1].length  // 3  
array[2].length  // 3  
array[3].length  // 3
```

توجه: دسترسی به اندیس خارج از محدوده مانند `array[4].length` باعث خطای `ArrayIndexOutOfBoundsException` می‌شود.

جمع دو ماتریس

صورت مسئله: دو ماتریس همابعاد داریم (3×2). هدف این است که ماتریس حاصل جمع را با جمع عنصر به عنصر به دست بیاوریم؛ یعنی هر خانه از ماتریس خروجی برابر مجموع مقدار متناظر در ماتریس‌های ورودی باشد. در نهایت، مقادیر ماتریس جمع به صورت سطر به سطر چاپ شوند.

```
class MatrixAddition {
    public static void main(String[] args) {
        // define two input matrices (2 rows, 3 columns)
        int[][] a = { {1, 3, 4}, {3, 4, 5} };
        int[][] b = { {1, 3, 4}, {3, 4, 5} };

        // result matrix to store element-wise sum
        int[][] c = new int[2][3];

        // compute sum and print row by row
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 3; j++) {
                c[i][j] = a[i][j] + b[i][j];
                System.out.print(c[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

2	6	8
6	8	10

آرایه‌های نامتقارن (Ragged Arrays)

در زبان‌های مانند جاوا، آرایه‌های دو بعدی در واقع آرایه‌ای از آرایه‌ها هستند. این یعنی هر سطر یک آرایه مستقل است و می‌تواند طول متفاوتی داشته باشد. به چنین ساختاری آرایه نامتقارن گفته می‌شود. این ویژگی باعث انعطاف‌پذیری بیشتر نسبت به آرایه‌های منظم می‌شود و امکان مدل‌سازی داده‌هایی با ابعاد نامنظم را فراهم می‌کند.

ویژگی‌های کلیدی:

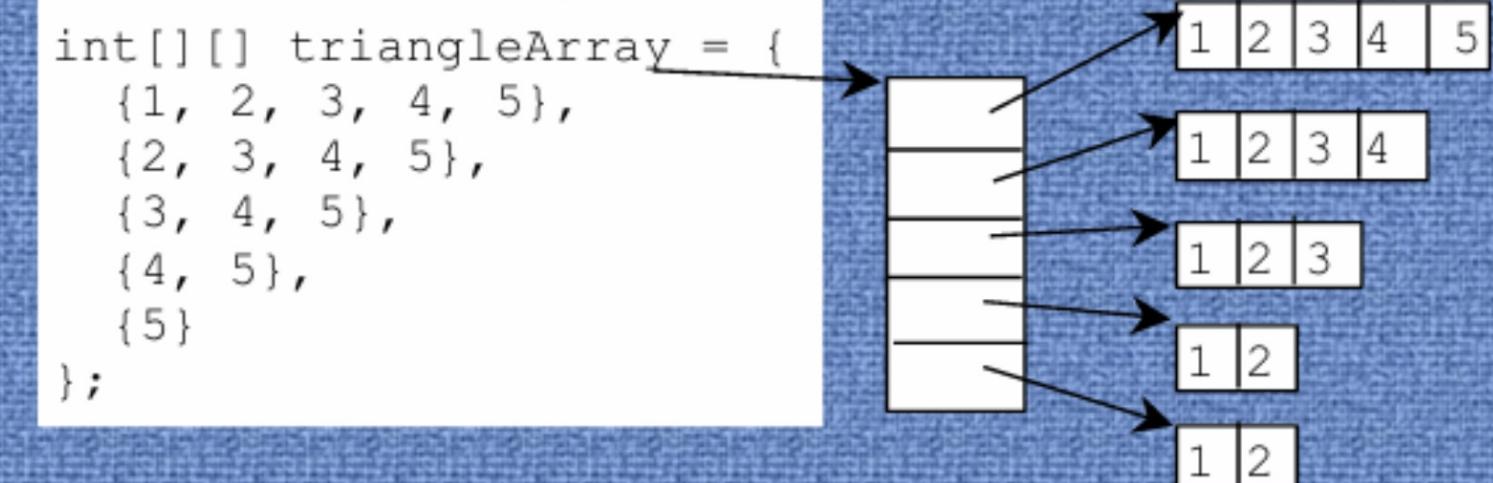
- هر سطر می‌تواند طول متفاوتی داشته باشد.
- حافظه بهینه‌تر مصرف می‌شود، چون فقط به اندازه‌ی موردنیاز هر سطر حافظه تخصیص داده می‌شود.
- برای نمایش ساختارهایی مثل مثلث، داده‌های نامنظم یا ماتریس‌های با طول متغیر بسیار مناسب است.

مثال :

```
int[][] matrix = {
    {1, 2, 3, 4, 5},
    {2, 3, 4, 5},
    {3, 4, 5},
    {4, 5},
    {5}
};
matrix.length      // 5
matrix[0].length  // 5
matrix[1].length  // 4
matrix[2].length  // 3
matrix[3].length  // 2
matrix[4].length  // 1
```

: مثال

```
int[][] randomArray = {  
    {10, 20},  
    {5, 15, 25, 35},  
    {7},  
    {100, 200, 300},  
    {42, 43, 44, 45, 46, 47}  
};
```



پردازش آرایه‌های چندبعدی

آرایه‌های چندبعدی می‌توانند با روش‌های مختلف پردازش شوند. این پردازش‌ها شامل مقداردهی اولیه، پیمایش، محاسبات و تغییر ساختار داده‌ها است. در ادامه مثالهایی از مهم‌ترین عملیات‌های رایج آورده شده‌اند:

1. مقداردهی اولیه با ورودی‌ها (Initializing arrays with input values)

2. چاپ عناصر آرایه (Printing arrays)

3. محاسبه مجموع همه عناصر (Summing all elements)

4. محاسبه مجموع عناصر به صورت ستونی (Summing all elements by column)

5. یافتن سطری که بیشترین مجموع را دارد (Which row has the largest sum)

6. یافتن کوچک‌ترین اندیس عنصر بیشینه (Finding the smallest index of the largest element)

7. جابجایی تصادفی عناصر (Random shuffling)

مقداردهی آرایه‌ها با داده‌های ورودی کاربر

صورت مسئله: هدف این است که مقادیر یک آرایه دو بعدی از کاربر دریافت شود. ابتدا ابعاد آرایه مشخص است و سپس با استفاده از حلقه‌های تو در تو، مقادیر هر خانه از طریق ورودی کاربر پر می‌شود.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int[][] matrix = new int[3][3];

        Scanner input = new Scanner(System.in);
        System.out.println("Enter " + matrix.length + " rows and " + matrix[0].length + " columns: ");
        for (int row = 0; row < matrix.length; row++) {
            for (int column = 0; column < matrix[row].length; column++) {
                matrix[row][column] = input.nextInt();
            }
        }

        input.close();
    }
}
```

مقداردهی آرایه‌ها با داده‌های تصادفی

صورت مسئله: هدف این است که یک آرایه دو بعدی با مقادیر تصادفی پر شود. برای این کار از حلقه‌های تو در تو استفاده می‌کنیم و در هر خانه یک عدد صحیح تصادفی بین ۰ تا ۹۹ قرار می‌دهیم.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int[][] matrix = new int[3][3];

        for (int row = 0; row < matrix.length; row++) {
            for (int column = 0; column < matrix[row].length; column++) {
                matrix[row][column] = (int)(Math.random() * 100);
            }
        }

        for (int[] rows : matrix) {
            for (int value : rows) {
                System.out.print(value + " ");
            }
            System.out.println();
        }
    }

    Main.main(null);
}
```

9	13	25
15	10	79
16	55	51

چاپ آرایه‌ها

صورت مسئله: هدف این است که عناصر یک آرایه دو بعدی به صورت سطر به سطر چاپ شوند. برای این کار از دو حلقه تو در تو استفاده می‌کنیم: حلقه بیرونی برای پیمایش سطرها و حلقه درونی برای پیمایش ستون‌های هر سطر. پس از چاپ هر سطر، یک خط جدید ایجاد می‌شود.

```
public static void main(String[] args) {
    int[][] matrix = new int[3][3];

    for (int row = 0; row < matrix.length; row++) {
        for (int column = 0; column < matrix[row].length; column++) {
            matrix[row][column] = (int)(Math.random() * 100);
        }
    }
    for (int row = 0; row < matrix.length; row++) {
        for (int column = 0; column < matrix[row].length; column++) {
            System.out.print(matrix[row][column] + " ");
        }
        System.out.println();
    }
}
```

استفاده از Enhanced loop در آرایه‌های دو بعدی

صورت مسئله: هدف این است که با استفاده از حلقه‌ی پیشرفته (for-each) عناصر یک آرایه دو بعدی پیمایش شوند. در این روش، به جای استفاده از اندیس‌ها، مستقیماً روی آرایه‌های داخلی و سپس عناصر آن‌ها حرکت می‌کنیم.

```
public class EnhancedLoop
{
    public static void main(String[] args) {
        int Site[][] = new int[10][10];
        for (int[] i : Site)
        {
            for (int j : i){
                i[j] = 1;
                System.out.print(" " + i[j]);
            }
            System.out.println();
        }
    }
}
```

مجموع همه عناصر آرایه

صورت مسئله: هدف این است که مجموع تمام عناصر یک آرایه دو بعدی محاسبه شود. برای این کار از دو حلقه تو در تو استفاده می کنیم: حلقه بیرونی برای پیمایش سطرها و حلقه درونی برای پیمایش ستون های هر سطر.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int[][] matrix = new int[3][3];

        for (int row = 0; row < matrix.length; row++) {
            for (int column = 0; column < matrix[row].length; column++) {
                matrix[row][column] = (int)(Math.random() * 100);
            }
        }

        int total = 0;
        for (int row = 0; row < matrix.length; row++) {
            for (int column = 0; column < matrix[row].length; column++) {
                total += matrix[row][column];
            }
        }

        System.out.println("Total = " + total);
    }
}
```

محاسبه مجموع عناصر هر ستون

صورت مسئله: هدف این است که مجموع عناصر هر ستون از یک آرایه دو بعدی محاسبه شود. برای این کار، ابتدا روی ستون‌ها پیمایش می‌کنیم و سپس در هر ستون با یک حلقه داخلی، عناصر آن ستون را جمع می‌زنیم. در پایان، مجموع هر ستون چاپ می‌شود.

```
public class Main {
    public static void main(String[] args) {
        int[][] matrix = new int[3][3];
        for (int row = 0; row < matrix.length; row++) {
            for (int column = 0; column < matrix[row].length; column++) {
                matrix[row][column] = (int)(Math.random() * 100);
            }
        }
        for (int column = 0; column < matrix[0].length; column++) {
            int sum = 0;
            for (int row = 0; row < matrix.length; row++) {
                sum += matrix[row][column];
            }
            System.out.println("Sum for column " + column + " is " + sum);
        }
    }
}
```

برزدن تصادفی (شافلینگ)

صورت مسئله: هدف این است که عناصر یک آرایه دو بعدی به صورت تصادفی جا به جا شوند. برای این کار، هر عنصر با یک عنصر تصادفی دیگر در آرایه تعویض می شود. این روش باعث می شود ترتیب عناصر به طور کامل بهم بخورد.

```
public class Main {
    public static void main(String[] args) {
        int[][] matrix = new int[3][3];
        for (int row = 0; row < matrix.length; row++) {
            for (int column = 0; column < matrix[row].length; column++) {
                matrix[row][column] = (int)(Math.random() * 100);
            }
        }
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                int i1 = (int)(Math.random() * matrix.length);
                int j1 = (int)(Math.random() * matrix[i1].length);

                int temp = matrix[i][j];
                matrix[i][j] = matrix[i1][j1];
                matrix[i1][j1] = temp;
            }
        }
        System.out.println("\nAfter Shuffle:");
        for (int[] row : matrix) {
            for (int value : row) {
                System.out.print(value + " ");
            }
            System.out.println();
        }
    }
}
```

ارسال آرایه‌های دو بعدی به متدها

صورت مسئله: هدف این است که نشان دهیم چگونه می‌توان یک آرایه دو بعدی را به عنوان آرگومان به یک متدها ارسال کرد. در این مثال، ابتدا مقادیر آرایه از کاربر دریافت می‌شوند، سپس آرایه به متدهای به نام `sum` ارسال می‌شود تا مجموع همه عناصر آن محاسبه و برگردانده شود.

```
import java.util.Scanner;

public class PassTwoDimensionalArray {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Enter array values
        int[][] m = new int[3][4];
        System.out.println("Enter " + m.length + " rows and " + m[0].length + " columns: ");
        for (int row = 0; row < m.length; row++) {
            for (int column = 0; column < m[row].length; column++)
                m[row][column] = input.nextInt();

        // Display result
        System.out.println("\nSum of all elements is " + sum(m));
    }

    public static int sum(int[][] m) {
        int total = 0;
        for (int row = 0; row < m.length; row++) {
            for (int column = 0; column < m[row].length; column++)
                total += m[row][column];

        return total;
    }
}
```

تصحیح خودکار تست چند گزینه‌ای با آرایه‌های دوبعدی

در این مثال، با استفاده از آرایه‌های دو بعدی، پاسخ‌های دانشآموزان به یک آزمون چند گزینه‌ای بررسی می‌شود. هر دانشآموز یک آرایه از پاسخ‌ها دارد و با مقایسه آن با کلید صحیح، نمره نهایی محاسبه می‌شود.

مراحل نوشتمن برنامه:

تعریف آرایه دو بعدی برای پاسخ‌های دانشآموزان
تعریف آرایه یک بعدی برای کلید صحیح
پیمایش هر دانشآموز و مقایسه پاسخ‌ها
شمارش تعداد پاسخ‌های صحیح
چاپ نمره نهایی هر دانشآموز

Key to the Questions:									
0	1	2	3	4	5	6	7	8	9
Key									
D	B	D	C	C	D	A	E	A	D

Students' Answers to the Questions:									
0	1	2	3	4	5	6	7	8	9
A	B	A	C	C	D	E	E	A	D
D	B	A	B	C	A	E	E	A	D
E	D	D	A	C	B	E	E	A	D
C	B	A	E	D	C	E	E	A	D
A	B	D	C	C	D	E	E	A	D
B	B	E	C	C	D	E	E	A	D
B	B	A	C	C	D	E	E	A	D
E	B	E	C	C	D	E	E	A	D

```
public class GradeExam {
    public static void main(String[] args) {
        char[][] answers = {
            {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
            {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'},
            {'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
            {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
            {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
            {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
            {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
            {'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'}
        };
        char[] keys = {'B', 'D', 'D', 'C', 'A', 'C', 'B', 'C', 'D', 'A'};
        for (int i = 0; i < answers.length; i++) {
            int correctCount = 0;
            for (int j = 0; j < answers[i].length; j++) {
                if (answers[i][j] == keys[j])
                    correctCount++;
            }
            System.out.println("Student " + i + "'s correct count is " + correctCount);
        }
    }
}
```

یافتن نزدیکترین دو نقطه در صفحه مختصات

در این مسئله، مجموعه‌ای از نقاط در صفحه مختصات داده شده‌اند. هدف، یافتن دو نقطه‌ای است که کمترین فاصله اقلیدسی را نسبت به یکدیگر دارند.

y	x	ردیف
۳	۱-	۶
۱-	۱-	۱
۱	۱	۲
۰,۵	۲	۳
۱-	۲	۴
۳	۳	۵
۰,۵-	۴	۶

برای یافتن نزدیکترین دو نقطه، می‌توان فاصله بین هر جفت را با فرمول اقلیدسی محاسبه کرد: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. سپس کمترین مقدار را انتخاب کرد.

```
import java.util.Scanner;

public class FindNearestPoints {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt for number of points
        System.out.print("Enter the number of points: ");
        int numberOfPoints = input.nextInt();

        // Create array to store points
        double[][] points = new double[numberOfPoints][2];
        System.out.println("Enter " + numberOfPoints + " points:");
        for (int i = 0; i < numberOfPoints; i++) {
            points[i][0] = input.nextDouble(); // x-coordinate
            points[i][1] = input.nextDouble(); // y-coordinate
        }

        // Initialize with first two points
        int p1 = 0, p2 = 1;
        double shortestDistance = distance(points[p1][0], points[p1][1],
                                           points[p2][0], points[p2][1]);
```

```
// Compare all pairs to find the closest
for (int i = 0; i < numberOfPoints; i++) {
    for (int j = i + 1; j < numberOfPoints; j++) {
        double d = distance(points[i][0], points[i][1],
                             points[j][0], points[j][1]);

        if (d < shortestDistance) {
            p1 = i;
            p2 = j;
            shortestDistance = d;
        }
    }
}

System.out.println("The closest two points are: (" +
                   points[p1][0] + ", " + points[p1][1] + ") and (" +
                   points[p2][0] + ", " + points[p2][1] + ")");

// Method to compute Euclidean distance
public static double distance(double x1, double y1, double x2, double y2) {
    return Math.sqrt((x2 - x1) * (x2 - x1) +
                    (y2 - y1) * (y2 - y1));
}
```

آرایه‌های چندبعدی

در برخی مسائل، نیاز داریم داده‌ها را در ساختارهایی پیچیده‌تر از آرایه‌های یکبعدی ذخیره کنیم. آرایه‌های چندبعدی این امکان را فراهم می‌کنند تا داده‌ها را در قالب ماتریس‌ها، مکعب‌های عددی یا ساختارهای n بعدی سازماندهی کنیم.

در زبان جاوا، می‌توان آرایه‌هایی با بیش از دو بعد تعریف کرد. ساختار تعریف آن مشابه آرایه‌های دوبعدی است، با این تفاوت که تعداد براکت‌ها و اندازه‌ها بیشتر می‌شود.

مثال: تعریف آرایه سهبعدی

در مثال زیر، آرایه‌ای سهبعدی از نوع **double** تعریف شده است که می‌تواند داده‌هایی در قالب $10 \times 5 \times 2$ ذخیره کند:

```
double[][][] scores = new double[10][5][2];
```

برای دسترسی به یک مقدار خاص در این آرایه، باید سه اشاره مشخص کنیم. مثلًاً عبارت `scores[3][2][1]` به مقداری اشاره دارد که: در ردیف چهارم (از ۱۰ ردیف)، در سطر سوم (از ۵ سطر)، و در ستون دوم (از ۲ ستون) قرار دارد.

مسئله: محاسبه مجموع امتیازها

می خواهیم برنامه‌ای بنویسیم که مجموع امتیازها را برای دانشجویان یک کلاس محاسبه می‌کند. فرض کنید امتیازها در یک آرایه [سه بعدی](#) به نام [scores](#) ذخیره شده‌اند. اندیس اول [دانشجو](#) را مشخص می‌کند، اندیس دوم [امتحان](#) را، و اندیس سوم [بخش امتحان](#) را. فرض کنید ۷ دانشجو و ۵ امتحان داریم و هر امتحان دو بخش دارد: یک بخش [چندگزینه‌ای](#) و یک بخش [تشریحی](#). بنابراین [\[0\]\[j\]\[0\]](#) بیانگر امتیاز دانشجوی [نام](#) در امتحان [زم](#) در بخش [چندگزینه‌ای](#) می‌باشد. برنامه‌ی زیر امتیاز کل را برای هر دانشجو نمایش می‌دهد.

```
public class TotalScore {
    /* Main method */
    public static void main(String args[]) {
        double[][][] scores = {
            {{7.5, 20.5}, {9.5, 22.5}, {15, 33.5}, {13, 21.5}, {15, 21.5}},
            {{5.5, 20.5}, {9.5, 22.5}, {15, 33.5}, {13, 21.5}, {15, 21.5}},
            {{6.5, 20.5}, {9.5, 22.5}, {15, 33.5}, {13, 21.5}, {15, 21.5}},
            {{6.5, 20.5}, {9.5, 22.5}, {15, 33.5}, {13, 21.5}, {15, 21.5}},
            {{6.5, 20.5}, {9.5, 22.5}, {15, 33.5}, {13, 21.5}, {15, 21.5}},
            {{6.5, 20.5}, {9.5, 22.5}, {15, 33.5}, {13, 21.5}, {15, 21.5}},
            {{6.5, 20.5}, {9.5, 22.5}, {15, 33.5}, {13, 21.5}, {15, 21.5}},
            {{6.5, 20.5}, {9.5, 22.5}, {15, 33.5}, {13, 21.5}, {15, 21.5}},
            {{6.5, 20.5}, {9.5, 22.5}, {15, 33.5}, {13, 21.5}, {15, 21.5}},
            {{1.5, 29.5}, {6.4, 22.5}, {10.5, 26.5}, {10, 30.5}, {16, 6.0}}
        };

        // Calculate and display total score for each student
        for (int i = 0; i < scores.length; i++) {
            double totalScore = 0;
            for (int j = 0; j < scores[i].length; j++) {
                for (int k = 0; k < scores[i][j].length; k++)
                    totalScore += scores[i][j][k];
            }
            System.out.println("Student " + i + "'s score is " + totalScore);
        }
    }
}
```

ساختار لیست متغیر آرگومان‌ها (Varargs)

ساختار **varargs** (لیست متغیر آرگومان‌ها) به یک متدهای امکان می‌دهد صفر یا چند آرگومان ورودی بپذیرد.

در صورت عدم استفاده از **varargs** می‌توانیم از متدهای سربارگذاری شده استفاده کنیم یا یک **آرایه** را به عنوان پارامتر ارسال نماییم. اما این روش‌ها به علت مشکلات نگهداری، مناسب به حساب نمی‌آیند.

در مواقعی که ندانیم چه تعداد آرگومان را به یک متدهای ارسال کنیم، **varargs** روشی بسیار مناسب خواهد بود.

:varargs مزیت

- به متدهای سربارگذاری شده نیاز نداریم.
- میزان کدنویسی کاهش می‌یابد.
- انعطاف‌پذیری بیشتری در ارسال آرگومان‌ها خواهیم داشت.

قاعده نحوی: از سه نقطه پشت سر هم ... پس از نوشتن نوع داده استفاده می‌کند:

```
//return_type method_name(data_type...VariableName){}
public static int sum(int... numbers) {
    int total = 0;
    for (int n : numbers) {
        total += n;
    }
    return total;
}
public static void main(String[] args) {
    System.out.println(sum());           // 0
    System.out.println(sum(5));          // 5
    System.out.println(sum(1, 2, 3, 4, 5)); // 15
}
```

همان‌طور که می‌بینید، متدهای `sum` می‌تواند بدون آرگومان، با یک آرگومان یا با چندین آرگومان فراخوانی شود، بدون نیاز به تعریف چندین متدهای سربرابر گذاری شده.

قواعد استفاده از `:varargs`

- بیش از یک آرگومان متغیر (`varargs`) در متدهای وجود نداشته باشد.
- در صورت استفاده از چند آرگومان، آرگومان متغیر (`varargs`) باید آخرین آرگومان باشد.

مثال ساده از varargs

در این مثال، متدهی به نام `display` تعریف شده که می‌تواند تعداد دلخواهی از رشته‌ها (`String`) را دریافت کند. این ویژگی با استفاده از ساختار `varargs` پیاده‌سازی شده است.

فراخوانی این متدهی می‌تواند بدون آرگومان یا با چندین آرگومان انجام شود، بدون نیاز به تعریف چندین نسخه از متدهی (Overloading).

در این مثال، متدهی `display` دو بار فراخوانی شده است: یک بار بدون هیچ آرگومانی، و یک بار با چهار رشته.

```
class VarargsExample1 {  
    static void display(String... values) {  
        System.out.println("display method invoked");  
    }  
  
    public static void main(String args[]) {  
        display(); // zero argument  
        display("my", "name", "is", "varargs"); // four arguments  }}}
```

مثالی دیگر از varargs

در این مثال، متدهی به نام `display` تعریف شده که می‌تواند تعداد دلخواهی از رشته‌ها (`String`) را دریافت کند. سپس با استفاده از حلقه-`for`-`each` هر رشته را به صورت جداگانه چاپ می‌کند.

این روش به ما اجازه می‌دهد بدون نیاز به تعریف چندین نسخه از متدها، آن را با تعداد دلخواهی از آرگومان‌ها فراخوانی کنیم و هر آرگومان را پردازش کنیم.

در این مثال، متدهی `display` سه بار فراخوانی شده است: یک بار بدون آرگومان، یک بار با یک آرگومان، و یک بار با چهار آرگومان.

```
class VarargsExample2 {  
    static void display(String... values) {  
        System.out.println("display method invoked");  
        for (String s : values) {  
            System.out.println(s);  
        }  
    }  
  
    public static void main(String args[]) {  
        display(); // zero argument  
        display("hello"); // one argument  
        display("my", "name", "is", "varargs"); // four arguments  
    }  
}
```

خروجی این برنامه به صورت زیر خواهد بود:

```
display method invoked
display method invoked
hello
display method invoked
my
name
is
varargs
```

نکات مهم در ترکیب varargs با آرگومان‌های دیگر:

- پارامتر varargs باید همیشه آخرین پارامتر در لیست باشد.
- می‌توان قبل از varargs، هر تعداد آرگومان معمولی تعریف کرد.
- در زمان فراخوانی، ابتدا آرگومان‌های معمولی و سپس لیست متغیر ارسال می‌شود.
- می‌توان از حلقه **for-each** برای پیمایش مقادیر استفاده کرد.
- در برنامه‌های محاسباتی در صورت ارسال هیچ آرگومانی، باید مراقب تقسیم بر صفر بود.

مثال از محاسبه میانگین با استفاده از varargs

در این مثال، متدهی به نام `average` تعریف شده که می‌تواند تعداد دلخواهی از مقادیر `double` را دریافت کرده و میانگین آنها را محاسبه کند. این قابلیت با استفاده از `varargs` پیاده‌سازی شده است.

استفاده از `varargs` باعث می‌شود بتوانیم متدهی `average` را با هر تعداد آرگومان فراخوانی کنیم، بدون نیاز به تعریف چندین نسخه از متدهای مختلف. در این مثال، چهار عدد تعریف شده‌اند و میانگین آنها در سه حالت مختلف محاسبه شده است: با دو عدد، سه عدد، و چهار عدد.

```
public class VarargsTest {
    // calculate average
    public static double average(double... numbers) {
        double total = 0.0;

        for (double d : numbers)
            total += d;

        return total / numbers.length;
    }

    public static void main(String[] args) {
        double d1 = 10.0;
        double d2 = 20.0;
        double d3 = 30.0;
        double d4 = 40.0;

        System.out.printf("Average of d1 and d2 is %.1f%n", average(d1, d2));
        System.out.printf("Average of d1, d2 and d3 is %.1f%n", average(d1, d2, d3));
        System.out.printf("Average of d1, d2, d3 and d4 is %.1f%n", average(d1, d2, d3, d4));
    }
}
```

آرگومان‌های خط فرمان در جاوا – متدهای main

یکی از قابلیت‌های مهم زبان جاوا، امکان ارسال داده به برنامه در لحظه اجرا از طریق **خط فرمان** (Command Line) است. این داده‌ها که به آن‌ها **آرگومان‌های خط فرمان** main ارسال شده و در آرایه‌ای از رشته‌ها به نام args ذخیره می‌شوند. این ویژگی به برنامه‌نویس اجازه می‌دهد تا رفتار برنامه را بر اساس ورودی‌های لحظه‌ای تغییر دهد، بدون نیاز به تغییر کد منبع.

تعریف استاندارد متدهای main در جاوا به صورت زیر است:

```
public static void main(String[] args)
```

در این تعریف:

- آرایه‌ای از رشته‌های ورودی String[] args می‌دارد.
- هر عنصر این آرایه یک آرگومان است که به صورت رشته‌ای ذخیره می‌شود، حتی اگر عدد باشد.
- تعداد آرگومان‌ها با استفاده از args.length قابل محاسبه است.

کاربردهای رایج آرگومان‌های خط فرمان:

- ارسال تنظیمات یا گزینه‌های اجرایی به برنامه (مثل حالت debug یا verbose).
- مشخص کردن مسیر فایل‌ها یا منابع خارجی.
- ورودی‌های عددی یا متنی برای پردازش در برنامه.
- ایجاد برنامه‌هایی با انعطاف‌پذیری بالا بدون نیاز به رابط کاربری گرافیکی.

مثال اجرا:

```
javac InitArray.java
java InitArray 5 0 4
// نتیجه در آرایه args:
args[0] = "5"
args[1] = "0"
args[2] = "4"
```

در این مثال، برنامه **InitArray** با سه آرگومان اجرا شده است. این آرگومان‌ها به ترتیب در عناصر آرایه **args** قرار می‌گیرند و برنامه می‌تواند از آن‌ها برای مقداردهی اولیه، تنظیمات یا پردازش استفاده کند.

نکاتی برای استفاده بهتر:

- برای تبدیل آرگومان‌ها به نوع عددی، از متدهایی مانند **Integer.parseInt(args[i])** استفاده کنید.
- قبل از استفاده از آرگومان‌ها، بررسی کنید که **args.length** به اندازه کافی باشد تا از خطاهای **ArrayIndexOutOfBoundsException** جلوگیری شود.

کلاس Arrays در جاوا

کلاس **Arrays** یکی از کلاس‌های کاربردی در بسته **java.util** است که مجموعه‌ای از متدهای استاتیک برای کار با آرایه‌ها فراهم می‌کند. این کلاس به شما اجازه می‌دهد بدون نیاز به نوشتن کدهای تکراری و پیچیده، عملیات رایج مانند مرتب‌سازی، جستجو، مقایسه و مقداردهی آرایه‌ها را انجام دهید. به عبارتی، **چرخ را دوباره اختراع نکنید!** از ابزارهای آمادهٔ جاوا استفاده کنید.

ویژگی‌های کلیدی کلاس Arrays

- دارای متدهایی برای مرتب‌سازی آرایه‌ها به صورت صعودی یا سفارشی با استفاده از مقایسه‌گرها.
- امکان جستجوی دودویی (**binary search**) برای یافتن عناصر در آرایه‌های مرتب.
- استفاده از **equals** برای مقایسه آرایه‌ها از نظر محتوا و ترتیب.
- متد **fill** برای مقداردهی تمام عناصر آرایه با یک مقدار مشخص.
- متدهای کلاس برای انواع داده‌های اولیه (...int, double, char) و اشیاء سربارگذاری شده‌اند.

```
import java.util.Arrays;
public class ArrayExample {
    public static void main(String[] args) {
        int[] numbers = {5, 2, 9, 1, 3};
        Arrays.sort(numbers); // مرتبسازی
        System.out.println("Sorted: " + Arrays.toString(numbers));

        int index = Arrays.binarySearch(numbers, 3); // جستجو
        System.out.println("Index of 3: " + index);

        int[] copy = Arrays.copyOf(numbers, numbers.length);
        System.out.println("Equal: " + Arrays.equals(numbers, copy)); // مقایسه

        Arrays.fill(copy, 0); // مقداردهی
        System.out.println("Filled: " + Arrays.toString(copy));
    }
}
```

نکاتی درباره این مثال :

- برای استفاده از کلاس **Arrays** نیازی به ساخت شیء نیست؛ تمام متدها **static** هستند.
- قبل از استفاده از **binarySearch**، آرایه باید مرتب شده باشد؛ در غیر این صورت نتیجه نامعتبر خواهد بود.
- متod **equals** فقط محتوا و ترتیب عناصر را بررسی می‌کند، نه مرجع حافظه.
- متod **toString** برای نمایش آرایه‌ها به صورت رشته‌ای بسیار مفید است.

ArrayList و کلاس Collections

زبان جاوا مجموعه‌ای از ساختارهای داده‌ی پیش‌ساخته تحت عنوان **Collections** ارائه می‌دهد که امکان ذخیره‌سازی، بازیابی و مدیریت گروهی از اشیاء را فراهم می‌کند. این ساختارها جایگزین‌های پیشرفته‌تری برای آرایه‌ها هستند و انعطاف‌پذیری بیشتری در مدیریت داده‌ها دارند.

ویژگی‌های کلیدی:

- امکان ذخیره‌سازی مجموعه‌ای از داده‌ها در یک Collection
- قابلیت بازیابی و تغییر داده‌ها
- پشتیبانی از جستجو، مرتب‌سازی و حذف داده‌ها
- افزایش بهره‌وری در مدیریت داده‌های پویا

یکی از کلاس‌های مهم در مجموعه‌ی Collections، کلاس **java.util.ArrayList<T>** از پکیج **java.util** است. این کلاس لیستی پویا از داده‌ها را فراهم می‌کند که می‌تواند در زمان اندازه‌ی خود را تغییر دهد. برخلاف آرایه‌های معمولی، **ArrayList** محدود به اندازه‌ی ثابت نیست و با افزودن یا حذف عناصر، اندازه‌ی آن به‌طور خودکار تنظیم می‌شود.

کلاس ArrayList و نحوه اعلان آن

کلاس **ArrayList<T>** در جاوا با استفاده از **نوع عمومی (Generic)** تعریف می‌شود. حرف **T** در این ساختار به عنوان **نگهدارنده مکان عمل می‌کند**: یعنی هنگام ایجاد یک شیء از **ArrayList**, نوع داده‌ای موردنظر را مشخص می‌کنیم. این رفتار مشابه آرایه‌های معمولی است که هنگام تعریف، نوع عناصرشان را تعیین می‌کنیم.

کلاس **ArrayList** معمولاً برای نگهداری **اشاره‌گرها (References)** به اشیاء استفاده می‌شود، نه خود داده‌های اولیه. این ویژگی باعث انعطاف‌پذیری بیشتر در مدیریت انواع مختلف داده‌ها می‌شود.

نحوه اعلان یک **ArrayList**:

در مثال زیر، یک لیست از نوع **String** تعریف شده که می‌تواند رشته‌ها را به صورت پویا ذخیره کند:

مثال :

```
import java.util.ArrayList;
ArrayList<String> names = new ArrayList<>();
names.add("Ali");
names.add("Sara");
names.add("Reza");
System.out.println(names.get(1));    // Sara
System.out.println(names.size());   // 3
names.remove(0);
System.out.println(names);          // [Sara, Reza]
```

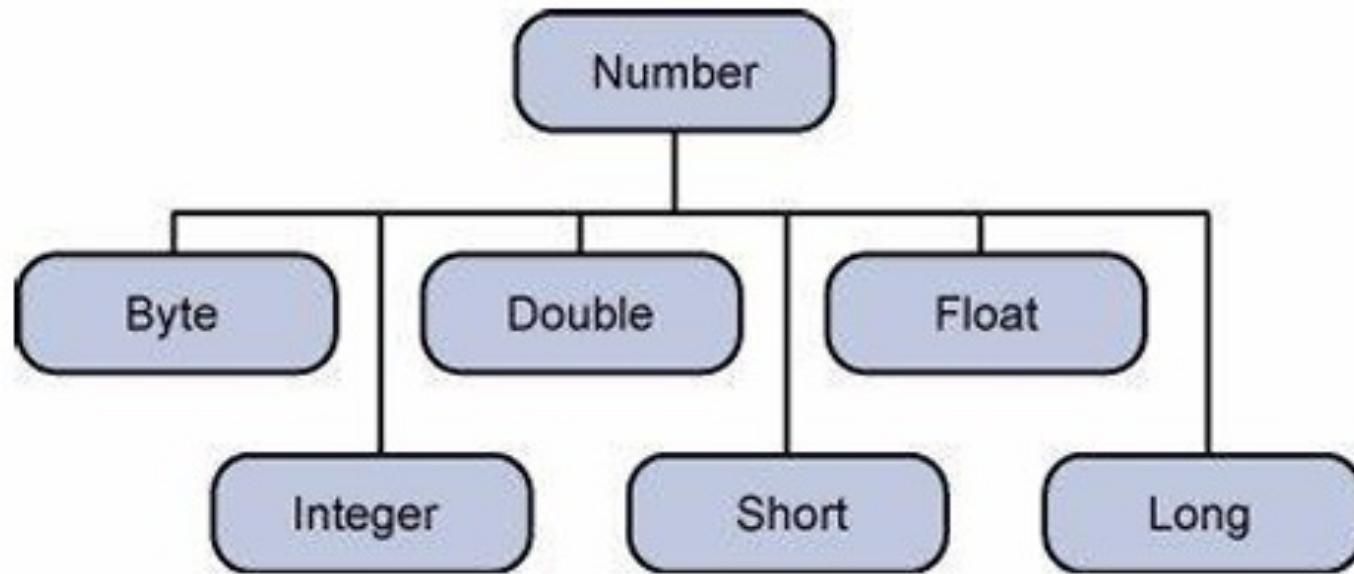
کلاس Number و wrapper های

از آنجایی که کلاس **ArrayList** در جاوا نمی‌تواند مقادیر از نوع‌های اصلی (primitive types) مانند **int** یا **double** را مستقیماً نگهداری کند، برای این منظور از **کلاس‌های wrapper** استفاده می‌شود. این کلاس‌ها نسخه‌ی شیء محور از نوع‌های اصلی هستند و همگی از یک کلاس انتزاعی به نام **Number** ارث‌بری می‌کنند.

کلاس‌هایی مانند **Short**, **Long**, **Byte** و **Integer**, **Double**, **Float** هستند و می‌توانند در ساختارهایی مانند **ArrayList** استفاده شوند. برای مثال، به جای ذخیره‌ی مقدار **int** در لیست، از کلاس **Integer** استفاده می‌کنیم.

: مثال

```
import java.util.ArrayList;
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(10);
numbers.add(25);
numbers.add(42);
System.out.println(numbers.get(2));    // 42
```



روش‌های پیمایش عناصر کلاس ArrayList

در حالت کلی به دو شیوه می‌توان به عناصر کلاس **ArrayList** و به طور خاص کلاس **Collection** دسترسی داشت:

- به صورت تکراری با تعریف واسط **Iterator** و فراخوانی متدهای **hasNext** و **next**
- با استفاده از ساختار حلقه **for-each**

مثال استفاده از **for-each**:

```
import java.util.*;
class TestCollection2{
    public static void main(String args[]){
        ArrayList<String> al = new ArrayList<String>();
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");
        for(String obj : al)
            System.out.println(obj);
    }
}
```

Ravi
Vijay
Ravi
Ajay

امكانات ArrayList

- 1 Make one

```
ArrayList<Egg> myList = new ArrayList<Egg>();
```

Don't worry about this new <Egg> angle-bracket syntax right now; it just means "make this a list of Egg objects".

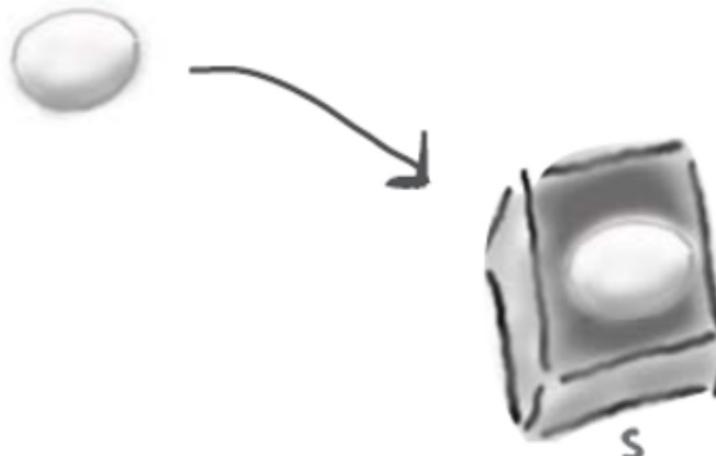


A new ArrayList object is created on the heap. It's little because it's empty.

- 2 Put something in it

```
Egg s = new Egg();
```

```
myList.add(s);
```

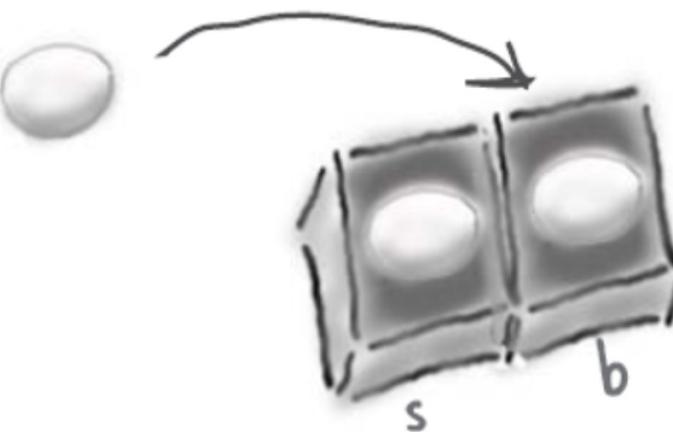


Now the ArrayList grows a "box" to hold the Egg object.

- ③ Put another thing in it

```
Egg b = new Egg();
```

```
myList.add(b);
```



The ArrayList grows again to hold the second Egg object.

- ④ Find out how many things are in it

```
int theSize = myList.size();
```

The ArrayList is holding 2 objects so the size() method returns 2

- ⑤ Find out if it contains something

```
boolean isIn = myList.contains(s);
```

The ArrayList DOES contain the Egg object referenced by 's', so contains() returns true

- ⑥ Find out where something is (i.e. its index)

```
int idx = myList.indexOf(b);
```

ArrayList is zero-based (means first index is 0)
and since the object referenced by 'b' was the
second thing in the list, **indexOf()** returns 1

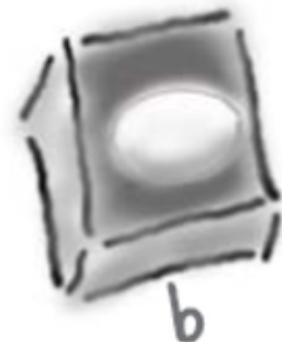
- ⑦ Find out if it's empty

```
boolean empty = myList.isEmpty();
```

it's definitely NOT empty, so **isEmpty()**
returns false

- ⑧ Remove something from it

```
myList.remove(s);
```



Hey look — it shrank!

مثال ArrayList با عناصر تعریف شده توسط کاربر

در این مثال، ابتدا کلاس **Student** تعریف می‌شود که شامل سه ویژگی است: شماره دانشآموز، نام و سن. سپس با استفاده از کلاس **ArrayList** مجموعه‌ای از دانشآموزان ساخته شده و اطلاعات آن‌ها چاپ می‌شود.

```
import java.util.*;
class Student {
    int rollno;
    String name;
    int age;
    Student(int rollno, String name, int age) {
        this.rollno = rollno;
        this.name = name;
        this.age = age;
    }
}
public class StudentList {
    public static void main(String args[]) {
        //Creating user-defined class objects
        Student s1 = new Student(101, "Sonoo", 23);
        Student s2 = new Student(102, "Ravi", 21);
        Student s3 = new Student(103, "Hanumat", 25);
        ArrayList<Student> list = new ArrayList<Student>();
        list.add(s1);
        list.add(s2);
        list.add(s3);
        Iterator<Student> itr = list.iterator();
        while (itr.hasNext()) {
            Student st = itr.next();
            System.out.println(st.rollno + " " + st.name + " " + st.age);
        }
    }
}
```

توجه :

با اینکه اشیای موجود در **Interface** از کلاس **Student** هستند، چون ریموت کنترل **itr** از نوع **Iterator** تعریف شده که یک عمومی است، نمی‌تواند به صفات ویژه شیء **Student** دسترسی پیدا کند.

در نتیجه، ابتدا باید یک **downcasting** انجام دهیم تا ریموت کنترل بتواند به همه فیلد‌های شیء **Student** دسترسی پیدا کند. این **downcasting** همواره به کلاس واقعی شیء موجود در **ArrayList** انجام شود تا بتوان از ویژگی‌های اختصاصی آن استفاده کرد.

نکته مهم دیگر این است که اگر در اعلان یک **ArrayList** از **placeholder** یا همان نوع **generic** استفاده نشود، به معنای تعریف مجموعه‌ای غیرژنریک یا سنتی خواهد بود. در این حالت، عناصر موجود در **ArrayList** از نوع کلاس **Object** خواهند بود.

اگر عناصر یک **ArrayList** از نوع **Object** باشند، می‌توانند اشیای مختلفی را در حافظه **heap** کنترل کنند. اما چون ریموت کنترل‌های ذخیره‌شده در همگی از نوع **Object** هستند، برای دسترسی به صفات ویژه اشیای کنترل شده توسط آنها، باید حتماً **downcasting** به کلاس اصلی هر شیء صورت گیرد.

ArrayList — Methods

Method	Return Type	Description	Example
ArrayList()		Creates an empty list.	ArrayList<String> list = new ArrayList<>();
add(Object)	boolean	Appends a new element to the end of the list.	list.add("Ali");
add(int, Object)	void	Inserts an element at the specified index.	list.add(1, "Sara");
clear()	void	Removes all elements from the list.	list.clear();
contains(Object)	boolean	Checks if the list contains the specified element.	list.contains("Ali");
get(int)	Object	Returns the element at the given index.	list.get(0);

indexOf(Object)	int	Returns the index of the first occurrence.	list.indexOf("Ali");
isEmpty()	boolean	Checks if the list is empty.	list.isEmpty();
lastIndexOf(Object)	int	Returns the index of the last occurrence.	list.lastIndexOf("Ali");
remove(int)	Object	Removes the element at the given index.	list.remove(2);
remove(Object)	boolean	Removes the specified element.	list.remove("Ali");
set(int, Object)	Object	Replaces the element at the given index.	list.set(0, "Reza");
size()	int	Returns the number of elements in the list.	list.size();

تفاوت آرایه معمولی و ArrayList

آرایه‌ها ساختارهایی با **اندازه ثابت** هستند که هنگام تعریف باید اندازه‌شان مشخص شود. اگر بخواهید مقدار یا شیئی را در آرایه قرار دهید، باید **اندیس دقیق** آن خانه را مشخص کنید. در صورت که **اندیس خارج از محدوده** تعریف شده باشد، با **خطای زمان اجرا** مواجه خواهید شد. (**ArrayIndexOutOfBoundsException**)

در مقابل، **ArrayList** یک ساختار پویا است که می‌تواند در طول اجرا **بزرگ یا کوچک شود**. هنگام ایجاد یک شیء از نوع **ArrayList**، نیازی به تعیین اندازه اولیه نیست. با استفاده از متدهای **add(index, object)** می‌توان عنصر جدیدی را به انتهای لیست اضافه کرد، یا با استفاده از **add(object)** عنصر را در موقعیت دلخواه قرار داد.

ArrayList	آرایه معمولی	ویژگی
در زمان اجرا قابل تغییر	در زمان تعریف	تعیین اندازه
با add(index, object) یا add()	با اندیس مشخص	افزودن عنصر
افزایش خودکار اندازه یا کنترل بهتر	در صورت اندیس اشتباه، خطای زمان اجرا	خطای اندیس
نوع‌های wrapper (مثلاً Integer)	نوع‌های اصلی (مثلاً int)	نوع داده

مثال :

```
// آرایه معمولی با اندازه ثابت
String[] names = new String[2];
names[0] = "Ali";
names[1] = "Sara";
// names[2] = "Reza"; // اندیس خارج از محدوده
// لیست پویا با ArrayList
ArrayList<String> list = new ArrayList<>();
list.add("Ali");
list.add("Sara");
list.add("Reza"); // نیاز به اندیس اضافه شدن بدون
System.out.println(list.get(2)); // Output: Reza
```

نمونه خطا در آرایه :

```
String[] arr = new String[2];
arr[5] = "Error"; // java.lang.ArrayIndexOutOfBoundsException
```

خودمون رو بسنجیم

این بخش برای این طراحی شده که در پایان مطالعه این اسلاید، بتونی خودت رو محک بزنی و ببینی آیا مفاهیم رو به خوبی یاد گرفتی یا نه. سوالات زیر را مرور کن و سعی کن بدون نگاه کردن به متن درس، به اون ها پاسخ بدی.

- متدى بنويسيد که عناصر هر سطر يك آرایه دو بعدی را جمع کند و حاصل را چاپ کند .
- برنامه اي بنويسيد که دو ArrayList را از ورودی دریافت و اشتراک آن دو را به انتهای دوم اضافه کند.
- روش های مختلف اضافه کردن عنصر به ArrayList را توضیح دهید

پایان

در صورت هرگونه سوال یا پیشنهاد میتوانید با من
در ارتباط باشید :)

gmail: nimasoltani.ce@gmail.com
telegram: @ni_s11