



## برنامه‌سازی پیشرفته و کارگاه

# آرایه و ArrayList و String ، کلاس HashMap

نگارش:

پریا قره خانی، نگین موسی بی جو، امیرمحمد ذاکر، کیانا پهلوان

استاد درس:

دکتر روح الله احمدیان

دانشکده ریاضی و علوم کامپیوتر

دانشگاه صنعتی امیرکبیر

ترم دوم ۱۴۰۴-۱۴۰۵

# فهرست مطالب

۱	آرایه و HashMap ، کلاس ArrayList و String	
۲	مقدمه	۱.۱
۲	آرایه - آرایه	۲.۱
۲	تعریف آرایه در جاوا	۱.۲.۱
۳	دسترسی به عناصر آرایه	۲.۲.۱
۳	پیمایش بر روی آرایه	۳.۲.۱
۴	متدها و فیلدهای کاربردی مرتبط با آرایه	۴.۲.۱
۵	آرایه‌های چندبعدی	۳.۱
۶	تعریف آرایه چند بعدی در جاوا	۱.۳.۱
۷	دسترسی به عناصر آرایه‌های چند بعدی	۲.۳.۱
۷	پیمایش آرایه‌های چند بعدی	۳.۳.۱
۸	ArrayList	۴.۱
۸	primitive type های معادل reference type در جاوا	۱.۴.۱
۹	ساخت اولین ArrayList	۲.۴.۱
۱۰	متدهای ArrayList	۳.۴.۱
۱۲	کلاس استرینگ در جاوا	۵.۱
۱۲	متدهای کلاس استرینگ	۱.۵.۱
۱۵	Wow! سوال	۲.۵.۱
۱۶	HashMap	۶.۱
۱۶	تعریف HashMap	۱.۶.۱
۱۷	کاربردهای HashMap	۲.۶.۱
۱۸	enhanced for loop	۷.۱
۱۹	چه چیزی یاد گرفتیم؟	۸.۱

# فصل ۱ آرایه و ArrayList ، کلاس HashMap و String

## ۱.۱ مقدمه

این داکیومنت برای آشنایی با چند ساختمان داده (Data Structure) آشنا در زبان جاوا می‌باشد، قرار است یاد بگیریم که این ساختمان داده‌ها به طور کلی به چه نحوی عمل می‌کنند و چه توابعی برای کارکردن با آن‌ها وجود دارد.

## ۲.۱ آرایه - Array

اگر بخواهید یک سری داده همنوع را ذخیره کنید، می‌توانید برای هر کدام به طور جداگانه یک متغیر درست کنید و مقداردهی کنید، اما این روش خیلی بهینه نیست! راه بهتر این است که از آرایه استفاده کنید. آرایه یک ساختار داده‌است که می‌تواند چندین مقدار همان‌نوع را داخل یک متغیر نگه دارد و مدیریت این داده‌ها را خیلی راحت‌تر می‌کند. شما مشابه این Data Structure را در زبان C هم دیدید. بباید ببینیم آرایه چه ویژگی‌هایی دارد و چطور کار می‌کند:

- اندیس گذاری (Indexing):** همان‌طور که احتمالاً از زبان C به یاد دارید، آرایه‌ها zero-based هستند، یعنی اولین عنصر آن‌ها اندیس ۰ دارد و آخرین عنصر آن‌ها اندیس  $(n - 1)$  خواهد داشت که  $n$  همان اندازه‌ی آرایه‌ست. مثلاً اگر یک آرایه ۵ تایی داشته باشیم، اندیس هایش از ۰ تا ۴ شماره‌گذاری می‌شوند.
- اندازه ثابت:** وقتی یک آرایه را می‌سازیم، اندازه‌اش همان موقع مشخص می‌شود و بعد از آن دیگر نمی‌توانیم آن را بزرگ‌تر یا کوچک‌تر کنیم.
- نوع داده یکسان:** تمامی خانه‌های یک آرایه باید از یک نوع داده باشند، مثلاً اگر نوع داده آرایه int باشد، نمی‌توانیم داخل آن String یا float بگذاریم. همهٔ اعضای آرایه باید با هم همان‌نوع باشند. حال بباید ببینیم که آرایه در جاوا به چه صورت کار می‌کند:

## ۱.۲.۱ تعریف آرایه در جاوا

حال می‌خواهیم آرایه را تعریف کنیم، این کار را می‌توانیم به دو روش انجام بدھیم:  
**تعریف آرایه بدون مقداردهی اولیه:** در این حالت فقط نوع داده‌ای که می‌خواهیم ذخیره کنیم و اندازه آرایه مشخص می‌شود:

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

در کد بالا، یک آرایه پنج عضوی تعریف کردیم، ولی مقادیر اولیه اعضای آن را مشخص نکردیم. با این کار، اعضای آرایه مقدار دیفالت خود را در اختیار می‌گیرند. مثلاً مقدار دیفالت یک متغیر int که مقداردهی نشده، صفر می‌باشد؛ به همین خاطر، اعضای آرایه بالا همگی صفر هستند. این موضوع را می‌توانید با چاپ این آرایه بررسی کنید:

```
for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + ", ");
}
```

خروجی کد بالا، به شکل زیر خواهد بود:

```
0, 0, 0, 0, 0,
```

مقادیر دیفالت‌های type عددی مثل int، float و غیره در جاوا صفر است. مقدار دیفالت reference type همان boolean بوده و مقدار دیفالت char هم '\n' می‌باشد؛ اما مقدار دیفالت ها همیشه null است که در مباحث مربوط به OOP راجع به آن‌ها مطالعه خواهید کرد.

**تعریف آرایه با مقداردهی اولیه:** در این حالت مقادیر آرایه همان موقع که تعریف می‌کنیم مشخص می‌شوند.

```
int[] numbers = {1, 2, 3};
```

## ۲.۲.۱ دسترسی به عناصر آرایه

بالاتر هم دیدید، در زبان C هم دیدید! برای دسترسی به عناصر یک آرایه، مثل کد زیر از index آن‌ها استفاده می‌کنیم:

```
void main(String[] args) {
    int[] numbers = {1, 2, 3};
    int firstNumber = numbers[0];
    numbers[2] = 25;

    System.out.println(numbers);
}
```

## ۳.۲.۱ پیمایش بر روی آرایه

حال بباید تک تک عناصر آرایه را با استفاده از اندیس و یک حلقه for چاپ کنیم.

```
void main(String[] args) {
    int[] numbers = {1, 2, 3};
    for (int i = 0; i < numbers.length; i++) {
        System.out.println("Index " + i + ": " + numbers[i]);
    }
}
```

ما، طول یک آرایه را با استفاده از `length` بدهست می‌آوریم. به متغیرهایی مثل `numbers` بدهست می‌گویند که از متغیرهای مربوط به آن کلاس می‌باشد. با این متغیرها و انواع آنها وقتی OOP را یاد گرفتید بیشتر آشنای خواهید شد. تا آن زمان، باید چندتا از متدها و فیلدهای پرکاربرد آرایه را بررسی کنیم:

## ۴.۲.۱ متدها و فیلدهای کاربردی مرتبط با آرایه

• متدهای `length`:

متدهای `length` طول آرایه را به شما نشان می‌دهد:

```
int[] numbers = {1, 2, 3};
System.out.println("Array length: " + numbers.length); // Array length: 3
```

• متدهای `System.arraycopy`:

برای کپی کردن آرایه‌ها می‌توانید از متدهای `System.arraycopy()` استفاده کنید، می‌توانید با استفاده از کد زیر، که آرایه `numbers` را داخل `copiedArray` کپی می‌کند، این متدهای را بررسی کنید:

```
int[] numbers = {1, 2, 3};
int[] copiedArray = new int[numbers.length];

System.arraycopy(numbers, 0, copiedArray, 0, numbers.length);

System.out.print("Copied array: ");
for (int i = 0; i < copiedArray.length; i++) {
    System.out.print(copiedArray[i] + " ");
}
```

خروجی این کد، به شکل زیر می‌باشد:

Copied array: 1 2 3

### • متدهای sort و Arrays.sort

با استفاده از متدهای sort و Arrays.sort آرایه‌ها را سورت کنید. قبل از استفاده از آن، توجه کنید که حتماً پکیج import (package) java.util.Arrays را کرده باشید:

```
import java.util.Arrays;

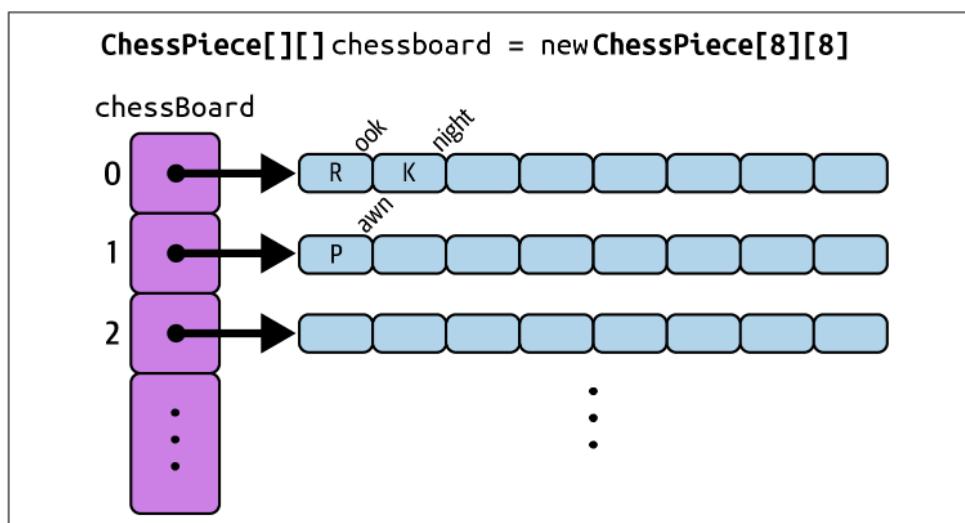
void main(String[] args) {
    int[] numbers = {5, 3, 8, 1, 2};
    System.out.println("Original array: " + Arrays.toString(numbers));

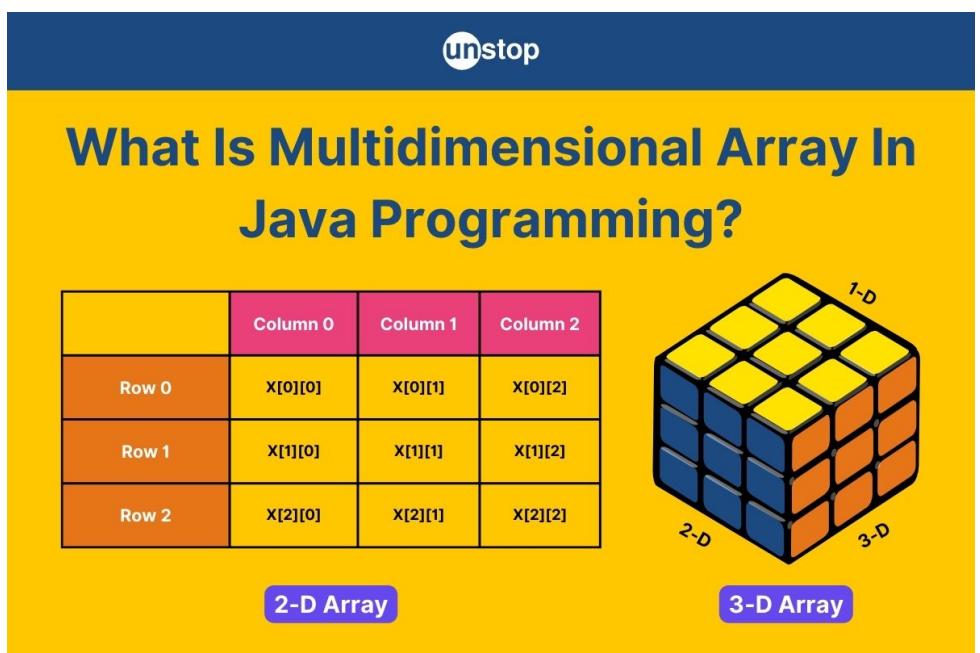
    Arrays.sort(numbers);
    System.out.println("Sorted array: " + Arrays.toString(numbers));
}
```

حال که آرایه‌یک بعدی را یاد گرفته ایم، یک مرحله بالاتر برویم و در مورد آرایه‌ی چند بعدی صحبت کنیم.

## ۳.۱ آرایه‌های چندبعدی

آرایه‌های چندبعدی برای ذخیره و مدیریت داده‌های پیچیده‌تر به کار می‌روند. تصویر پایین هم یک مثال خوب از آرایه‌ی دو بعدی را نشان می‌دهد:





در تصویر اول و سمت چپ تصویر دوم، شما یک آرایه دو بعدی می بینید که هر کدام از عناصر آرایه بیرونی، خود یک آرایه هستند. برای راحت تر فهمیدن، می توانید آرایه دو بعدی را مثل یک ماتریس در نظر بگیرید. در کنار آرایه دو بعدی، شما آرایه سه بعدی را می بینید که در واقع آرایه ای از آرایه های دو بعدی است؛ یعنی هر عنصر آن، خود یک آرایه دو بعدی است.

### ۱.۳.۱ تعریف آرایه چند بعدی در جاوا

در ابتدا، نحوه تعریف آرایه های دو بعدی و چند بعدی را تعریف می کنیم که مثل آرایه یک بعدی، به دو روش مختلف می شود انجامش داد.

**تعریف آرایه دو بعدی و چند بعدی بدون مقداردهی اولیه:** مثل قبل، می توانید بدون مقداردهی اولیه آرایه تعریف کنید و باز هم خانه های آرایه ها با مقدار دیفالت type شان پر می شوند.

```
// 2D array with 3 rows and 4 columns
int[][] matrix = new int[3][4];

// 3D array with 3 layers, each containing 3 rows and 3 columns
int[][][] cube = new int[3][3][3];

System.out.println(matrix[0][0]); // 0
System.out.println(cube[0][0][0]); // 0
```

**تعریف آرایه دو بعدی و چند بعدی با مقداردهی اولیه:** این روش هم، مشابه آرایه های یک بعدی هست:

```

int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

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

```

### ۲.۳.۱ دسترسی به عناصر آرایه‌های چند بعدی

مجدداً، با استفاده از اندیس‌ها مثل زبان C، می‌توانیم به عناصر آرایه‌های چند بعدی دست پیدا کنیم:

```

int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

int value1 = matrix[1][2];
matrix[0][1] = 42;

```

### ۳.۳.۱ پیمایش آرایه‌های چند بعدی

حال بباید تک تک عناصر آرایه را با استفاده از اندیس و یک حلقة for چاپ کنیم.

```

void main(String[] args) {
    int[][] matrix = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
}

```

```
};

System.out.println("Matrix contents:");
for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();
}
}
```

نهایتی که داخل این کد، نسبت به کدی که برای آرایه‌یک بعدی زدیم جدید است، متغیر  $z$  است که از صفر شروع می‌شود و تا  $length$   $matrix[i]$  پیش می‌رود. در واقع طول  $i$ -امین آرایه‌ای است که توابع  $matrix$  هست.

## ArrayList (f.)

تصور کنید که یک ساختمان داده داشته باشیم که شبیه آرایه باشد، ولی بتوانیم اندازه اش را هر موقع که خواستیم تغییر بدھیم. این ویژگی را `ArrayList` به ما می دهد. `ArrayList` سایز متغیری دارد که آن را بر اساس مقادیر داخلش مشخص می کند و استفاده از آن خیلی شبیه به یک `Array` معمولی است. حالا می خواهیم یک `ArrayList` بسازیم و سپس مهم ترین متدهایی که دارد را توضیح بدھیم.

۱.۴.۱ primitive type های معادل reference type ها در جاوا

برای هر primitive type توی جاوا، مثل int، boolean، char و امثال آن‌ها، یک معادل از جنس reference type وجود دارد. وقتی که با مفهوم object‌ها بیشتر آشنا بشویم، این موضوع را مجدداً بررسی می‌کنیم، ولی چون برای ArrayList به آن‌ها نیاز داریم، لازم است که یک نگاه مختصر به آن‌ها بیندازیم.

```
int num = 10;
```

ما م- تهانیم این متغیر را، به حای int، از حسوس Integer تعریف کنیم:

Integer num = 0;

تنها تفاوت این دو، این است که `Integer` reference type ای است که معادل `int` می باشد. کامپایلر شما حتی به شما این اجازه را می دهد که با متغیری از جنس `Integer` مثل یک `int` برخورد کنید:

```

int a = 10;
Integer b = a;
int c = a + b;

System.out.println(a);
System.out.println(b);

```

در مورد این که چرا وقتی int را داریم، به Integer هم نیاز داریم بعدا توضیح خواهیم داد. برای آن کافی است بدانید که این reference type ها وجود دارند و تا حد خیلی خوبی هم معادل همتای primitive type شان هستند. فهرست کامل آنها در جدول زیرآمده است:

Primitive Type	Reference Type (Wrapper Class)
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

در ArrayList ها، ما از این reference type ها استفاده می‌کنیم.

## ۲.۴.۱ ساخت اولین ArrayList

اول از همه باید package مربوط به ArrayList را import کنیم.

```
import java.util.*;
```

سپس، با تکه کد زیر می‌توانیم یک ArrayList از Integer ها و Float ها تعریف کنیم:

```

void main(String[] args) {
    ArrayList<Integer> intNumbers = new ArrayList<Integer>();
    ArrayList<Float> floatNumbers = new ArrayList<Float>();
}

```

دقت کنید که شما نمی‌توانید ArrayList ای از جنس primitive type تعریف کنید. به همین خاطر است که ما از Integer و Float استفاده کردیم. چیزی که بین < و > جلوی ArrayList می‌آید، جنس چیزی هست که داخل ArrayList نگه می‌دارید.

شما می‌توانید تایپ کامل ArrayList را مقابله دو از `new` نویسید و داخل <> را خالی بگذارید، چون که قبل از اسم متغیر، تایپ آن را کامل مشخص کردید:

```
ArrayList<Integer> numbers = new ArrayList<>();
```

یا این که به کل از var استفاده کنید:

```
var numbers = new ArrayList<Integer>();
```

## ۳.۴.۱ ArrayList متدهای

- متد add

برای اضافه کردن عناصر به ArrayList از این متد استفاده می‌کنیم. می‌توانید به دو شکل زیر به یک ArrayList عنصر اضافه کنید:

**اضافه کردن به آخر لیست:** کد زیر به خوبی به شمانشان می‌دهد که چگونه می‌توانید از ساده‌ترین نوع add استفاده کنید.

```
import java.util.ArrayList;

void main(String[] args) {
    ArrayList<Integer> numbers = new ArrayList<Integer>();

    numbers.add(10);
    numbers.add(20);
    numbers.add(30);
    numbers.add(40);

    System.out.println(numbers); // Output: [10, 20, 30, 40]
}
```

همچنین می‌توانید، به جای این که از چندتا add استفاده کنید تا مقداردهی اولیه آرایه‌تان را انجام بدهید، آن را به این شکل تعریف کنید:

```
ArrayList<Integer> numbers = new ArrayList<>(Arrays.asList(1, 3000, 40));
System.out.println(numbers); // Output: [1, 3000, 40]
```

با این کار، لازم نیست تا برنامه‌تان را پر از add بکنید.  
**اضافه کردن در یک ایندکس مشخص از لیست:** می‌توانید با استفاده از متد زیر، به یک جای مشخص از ArrayList یک عنصر را اضافه کنید.

```
ArrayList<Integer> numbers = new ArrayList<>(Arrays.asList(1, 3000, 40));
System.out.println(numbers); // Output: [1, 3000, 40]

numbers.add(1, 15);
System.out.println(numbers); // Output: [1, 15, 3000, 40]
```

remove متد .

می‌توانیم با ورودی دادن index عنصر دلخواه‌مان، آن را از آرایه پاک کنیم:

```
ArrayList<Character> letters = new ArrayList<>(Arrays.asList('A', 'B', 'C',
    'D', 'E', 'F'));
letters.remove(2); // Removes the third element of letters

System.out.println(letters); // Output: [A, B, D, E, F]
```

get متد .

این متد برای دسترسی به یک عنصر براساس index آن استفاده می‌شود و شبیه [i] ای است که در Array دیده بودیم:

```
var specialNumbers = new ArrayList<Double>(Arrays.asList(3.14, 2.71));
var pi = specialNumbers.get(0);

System.out.println(pi); // Output: 3.14
```

size متد .

این متد تعداد عناصر موجود در لیست را برمی‌گرداند.

```
var names = new ArrayList<String>(Arrays.asList("Negin", "Paria", "Kiana"));

for (int i = 0; i < names.size(); i++) {
    System.out.println(i + ". " + names.get(i));
}
```

set متد .

برای تغییر مقدار یک عنصر در یک ایندکس خاص از این متده استفاده می‌شود.

```
var names = new ArrayList<String>(Arrays.asList("Negin", "Paria", "Kiana"));
System.out.println(names); // Output: [Negin, Paria, Kiana]

names.set(1, "Dorsa");
System.out.println(names); // Output: [Negin, Dorsa, Kiana]
```

## ۵.۱ کلاس استرینگ در جاوا

اکنون می‌خواهیم با یکی از مهم‌ترین کلاس‌های جاوا آشنا شویم. استرینگ در جاوا آرایه‌ای از کارکترهاست که به صورت زیر و با " " تعریف می‌شود.

```
void main(String[] args) {
    String str = "write your string here";
}
```

### ۱.۵.۱ متدهای کلاس استرینگ

اکنون در کد خود یک رشته به صورت گفته شده تعریف کنید. در زیر چندین متده مهم از کلاس استرینگ گفته شده است. آن‌ها را بر روی رشته‌ی خود اجرا کنید و کاربردهای گوناگون آنها را ببینید.

• متده charAt .

فرض کنید می‌خواهیم به بخش‌های گوناگون یک رشته دسترسی یابیم. متده () برای انجام این کار به ما کمک می‌کند. این متده کارکتر موجود در ایندکس مشخص شده از رشته را برمی‌گرداند.

```
String str = "write your string here";
str.charAt(3); //returns t
```

• متده toCharArray .

این متده در هنگامی استفاده می‌شود که نیاز داریم روی هر کارکتر یک رشته عملیات خاصی انجام دهیم. این متده یک رشته را به آرایه‌ای از کارکترها تبدیل می‌کند.

```
String str = "Hello";
char[] charArray = str.toCharArray();
for (int i = 0; i < charArray.length-1; i++) {
```

```

        System.out.print(charArray[i] + ",");
    }
System.out.print(charArray[charArray.length-1]);

```

خروجی این کد H,e,l,o است.

• متد concat

به انتهای رشته، رشته مشخص شده را اضافه می‌کند.

```

String str = "Hello";
String name = " Negin";
str.concat(name); //returns Hello Negin

```

یک روش دیگر نیز برای افزودن رشته‌ها به یکدیگر وجود دارد:

```

String str = "Hello";
String name = " Paria";
String result = str + name;
System.out.println(result);

```

خروجی این کد Hello Paria است.

• متد contains

خروجی این متد Boolean است. بررسی می‌کند که آیا در درون رشته، رشته مشخص شده وجود دارد یا خیر. اگر وجود داشته باشد true و در غیر این صورت false را برمی‌گرداند.

```

String str = "Gharakhani's house";
str.contains("house");

```

• متد endsWith

متد دیگری که بررسی می‌کنیم متد endsWith() است. این متد بررسی می‌کند که آیا رشته با عبارت مشخص شده پایان یافته است یا خیر.

```

String str = "write your string here";
str.endsWith("here");

```

• متد `startsWith`

متد `()` مشابه متد قبلی است. بررسی می‌کند که آیا رشته با عبارت مشخص شده شروع یافته است یا خیر. مثلاً فرض کنید شما در برنامه از کاربران شماره تلفن شان را می‌خواهید. میتوانید با این متد بررسی کنید که شماره فرمات درستی دارد و با ۰۹۱ شروع شده است یا خیر.

• متد `equals`

دو رشته را باهم مقایسه می‌کند. اگر برابر بودند `true` را برمی‌گرداند و در غیر این صورت `false` را برمی‌گرداند.

```
String str = "AmirKabir University";
String sample= "University";
str.equals(sample); //returns false
str.equals("AmirKabir University"); //returns true
```

• متد `indexOf`

اولین جایگاه کارکتر مشخص شده در رشته را برمی‌گرداند. اگر کارکتر در استرینگ وجود نداشت -1 برمی‌گرداند.

```
String str = "write your string here";
str.indexOf('t'); //returns 3
str.indexOf('f'); //returns -1
```

• متد `length`

این متد بسیار کاربردی نیز، طول استرینگ داده شده را برمی‌گرداند.

```
String str = "write your string here";
str.length(); //returns 22
```

• متد `split`

این متد رشته را با استفاده از کارکتر داده شده به بخش‌های جداگانه تقسیم می‌کند و نتیجه را به صورت یک آرایه بازمی‌گرداند.

```
String str = "pizza,pasta,burger";
String[] arr = str.split(",");
System.out.println(arr[1]);
```

خروجی این کد pasta است.  
میتوانید کلماتی که میان آنها فاصله است را نیز با استفاده از این متدها کنید.

```
String str = "pizza pasta burger";
String[] arr = str.split(" ");
System.out.println(arr[1]);
```

خروجی این کد نیز pasta است.

#### • متدها

در پایان با یکی از کاربردی ترین متدهای کلاس استرینگ آشنا می‌شویم. این متدها از ایندکس اولیه داده شده (begIndex) تا قبل از ایندکس پایانی (endIndex) رشته را باز می‌گردانند. میتوانید را نگذارید و آنگاه تا آخر رشته برگردانده می‌شود. اکنون چند مثال از این متدها بینیم:

```
String str = "just keep swimming";
String result = str.substring(5,9);
System.out.println(result);
```

خروجی این کد keep است.  
اما اگر ایندکس پایانی را مانند مثال زیر نگذاریم:

```
String str = "just keep swimming";
String result = str.substring(5);
System.out.println(result);
```

در اینجا تا پایان رشته گرفته شده و keep swimming در خروجی چاپ می‌شود.  
حال با مهم ترین متدهای کلاس استرینگ آشنا شدید. این کلاس متدهای بسیار زیاد و گوناگونی دارد که در [این صفحه](#) می‌توانید با برخی دیگر از این متدها آشنا شوید.  
حال بباید یک سوال ساده را با استفاده از متدهایی که یاد گرفتیم حل کنیم.

## ۲.۵.۱ سوال Wow!

نگین که از کد زدن خسته شده است، به تازگی به رشته مهندسی برق علاقه پیدا کرده است. به همین دلیل تصمیم گرفته است تا درباره این رشته تحقیق کند. او به افراد مختلف مراجعه می‌کند و

هر کدام یک مقدار اطلاعات به او می‌دهند. او به اندازه‌ی مقدار اطلاعاتی که از هر فرد می‌گیرد، متعجب می‌شود. مثلاً اگر یک عدد اطلاعات بگیرد می‌گوید! Wow، اگر سه تا اطلاعات بگیرد می‌گوید! Wooow! به همین شکل تعداد ۵ ها زیاد می‌شود. حال اگر یک نفر به اندازه‌ی ۲۰ اطلاعات به نگین بدهد، ما باید انتظار چه کلمه‌ای را از او داشته باشیم؟

سعی کنید سوال بالا را با متدهایی که آموزش دیده‌اید حل کنید. برای مثال می‌توانید در هر مرحله با استفاده از متدهای `substring()` بخش ۷ ابتدایی را جدا کرده و در هر مرحله با استفاده از متدهای `concat()` یک ۵ به آن بیافزایید. این کار را  $n-1$  بار تکرار کنید. رشته‌ی نهایی پاسخ ما است.

## 6.1 HashMap

فرض کنید می‌خواهیم نام دانشجویان و شماره دانشجویی آن‌ها را در یک Data Structure ذخیره کنیم. می‌خواهیم به سرعت و راحتی به آن‌ها دسترسی داشته باشیم و در عین حال دانشجویان را از هم تفکیک کنیم. با توجه به آن که هر دانشجو یک شماره دانشجویی یکتا دارد، ما از یک Data Structure که از این ویژگی بهره می‌برد استفاده می‌کنیم. کلاس `HashMap` یکی از نمونه‌های خوب این گونه ساختمان داده‌هاست.

در این بخش، هدف ما آشنایی شما با مفهوم و کاربردهای کلاس `HashMap` است. دیکشنری یک نوع ساختار داده است که برای ذخیره کلید (key) و مقدار (value) استفاده می‌شود. برای مثال یک دفترچه تلفن را در نظر بگیرید. اسم افراد نقش key و شماره تلفن‌شان نقش value را دارد. مثلاً به برنامه می‌گویید که کلید Paria مقدار ۰۹۱۲۳۴۵۶۷۸۹ دارد. کلیدها یکتا (unique) هستند. مثلاً شما دو کلید Paria نمی‌توانید داشته باشید.

### 1.6.1 HashMap تعریف

در جاوا، به شکل زیر می‌توانید یک `HashMap` که جنس key های آن `String` و جنس value هایش `Integer` است تعریف کنید:

```
import java.util.HashMap;

void main(String[] args) {
    HashMap<String, Integer> studentId = new HashMap<>();
}
```

حال، می‌توانید با استفاده از متدهای `put` و `get` مقادیر دلخواه خود را اضافه کنید:

```
studentId.put("Negin", 13001);
studentId.put("Paria", 13002);
```

بعد از اجرای کد بالا، هشتم می‌باشد که عدد 13001 را برای کلید Negin و عدد 13002 را برای کلید Paria ذخیره کرده است. ما می‌توانیم با استفاده از متدهای `get` و `containsKey` دادن کلید دلخواه‌مان، مقدار متناظر با آن کلید را به دست بیاوریم:

```
var neginStudentId = studentId.get("Negin");
System.out.println(neginStudentId); // Output: 13001
```

اگر کلیدی در HashMap وجود نداشته باشد و ما آن را به عنوان ورودی به متدهیم، خروجی این متدهی برای ما null است. null ، مقدار خاصی برای reference type هاست که نشان می‌دهد که آن‌ها هنوز وجود ندارند و با آن در OOP بیشتر آشنا می‌شویم:

```
System.out.println(studentId.get("Yasin")); // Output: null
```

برای این که چک کنیم که هش مپ مان کلیدی را دارد یا نه، از containsKey استفاده می‌کنیم:

```
if (studentId.containsKey("Paria")) {
    System.out.println("Paria: " + studentId.get("Paria"));
} else {
    System.out.println("Cannot find Paria");
}
```

## ۲.۶.۱ کاربردهای HashMap

حال بیایید چند مثال و کاربرد از کلاس دیکشنری ببینیم تا به درک بهتری از آن برسیم.

- پیدا کردن سریع داده‌ها: میتوانید با دادن key خیلی سریع value آن را پیدا کنید. در درس ساختمان داده خواهید دید که این کار، بسیار سریع ترازگشتن در یک آرایه است:

```
HashMap<String, Integer> studentId = new HashMap<>();

studentId.put("Negin", 13001);
studentId.put("Paria", 13002);

var neginStudentId = studentId.get("Negin"); // neginStudentId = 13001
```

- شمارش تعداد دفعات تکرار کلمات: با هر بار دیدن کلمه در متن value تغییر داده شود و یک عدد به آن اضافه شود.

```
HashMap<String, Integer> studentCredit = new HashMap<>();

studentCredit.put("Negin", 0);
studentCredit.put("Paria", 0);

String creditsList = "Paria,Negin,Paria";

String[] gotCreditStudent = creditsList.split(",");
for (String studentName : gotCreditStudent){
    int temp = studentCredit.get(studentName);

    studentCredit.put(studentName, temp + 1);
}

System.out.println("Paria's Credit: " + studentCredit.get("Paria"));
System.out.println("Negin's Credit: " + studentCredit.get("Negin"));
```

حلقه for استفاده شده در اینجا به صورت enhanced for loop است که اگر اکنون با آن آشنایی ندارید پس از به پایان رساندن Document برگردید و این کد را دوباره مطالعه کنید.

- نگهداری ارتباط بین دو مجموعه داده: برای مثال key نام دانشآموز و value مجموعه نمراتش باشد.
- گروهبندی داده‌ها: برای مثال key الکترونیک و value لیست محصولات الکترونیک (لپ تاپ، تلفن همراه و ... ) باشد.

## enhanced for loop ۷.۱

حالا فرض کنید وقتی داریم یک آرایه یا هر ساختمان داده دیگه‌ای را پیمایش می‌کنیم و فقط به مقدار آن عنصر نیاز داریم و اصلاً کاری به ایندکسش نداریم. یعنی تمرکز ما بیشتر روی خود مقدار عنصر است تا ایندکسش. در این موقع می‌توانیم از enhanced for loop استفاده کنیم.

در واقع، دستور enhanced for loop همان حلقه for معمولی است که تعدادی تغییرات در ساختار آن ایجاد کردیم. یکی از مهم‌ترین تفاوت‌هایش این است که در enhanced for loop تمرکز بیشتر روی مقدار و ارزش هر عنصر است. این دستور برای پیمایش آرایه‌ها، ArrayList و ... استفاده می‌شود.

بگذارید با یک مثال بهتر توضیح دهیم:

```
void main(String[] args) {
    // An array of fruits
    String[] fruits = {"Apple", "Banana", "Orange"};
```

```
// Iterate over the array using for-each loop
for (String fruit : fruits) {
    // Print each fruit
    System.out.println(fruit);
}
```

در اینجا متغیر fruits آرایه‌ای از رشته‌ها است که در آن سه عنصر وجود دارد. این حلقه for جوری کار می‌کند که از اولین عنصر شروع می‌کند و یکی‌یکی تمام مقادارهای داخل آرایه را می‌گیرد. در هر دور از حلقه، مقدار fruit برابر با یکی از عناصر آرایه می‌شود (مثلًاً در دور اول Apple، دور دوم Banana و دور سوم Orange). بعد، مقدار fruit چاپ می‌شود. همان‌طور که دیدیم، توی این مدل for که به آن loop enhanced for می‌گویند، بیشتر روی مقدار عنصر تمرکز داریم تا ایندکس‌ش.

## ۸.۱ چه چیزی یاد گرفتیم؟

در این بخش آموختید که:

- آرایه‌ی ساده در جاوا چیست و چطور می‌توانیم از آن استفاده کنیم.
- آرایه‌ی چندبعدی چیست و چه کاربردهایی دارد.
- کلاس ArrayList در جاوا چیست، چه تفاوت‌هایی با آرایه‌ی ساده دارد، و متدهای اصلی آن (مانند add، remove، ...) چیست.
- کلاس String در جاوا برای ذخیره‌ی رشته‌ها چطور کار می‌کند و برخی از متدهای آن مثل charAt، equals، و ... چطور می‌توانند به ما کمک کنند.
- کلاس HashMap در جاوا چیست، و key و value چه معنایی دارند. همچنین باید با متدهای اصلی این کلاس مثل get و put آشنا باشید.
- حلقه‌های enhanced for در جاوا چه کاربردی دارند و بتوانید از آن برای پیمایش یک آرایه یا هش‌مپ استفاده کنید.

انتظار نمی‌رود که با این داکیومنت، از جزئیات پیاده‌سازی کلاس‌های String، ArrayList و HashMap و نحوه کار آن‌ها اطلاع داشته باشید. در هفته‌ی دهم این درس با این مفاهیم بیشتر آشنا می‌شویم و در درس «ساختمان داده‌ها و الگوریتم‌ها»، به صورت عمیق با این مباحث کار خواهید کرد. در حال حاضر، صریح استفاده از این ابزارها برای شما کافی است. اگر در هر یک از این مفاهیم مشکلی دارید، حتماً از تدریس‌یارها بپرسید. حتی اگر پرسش شما ساده باشد، حتماً بپرسید و از بازخورد تدریس‌یارها استفاده کنید.