آموزش زبان برنامه نویسی جاوا

کار با فایل در جاوا

Files and I/O

جلسه سی ام

نویسنده : رحمان زارعی

جاوا را ساده،آسان و شیرین بنوشید!!!!

در این جلسه آموزشی قصد داریم کار با فایل در جاوا را بررسی کنیم.

ما گاهی نیاز داریم داده ها و اطلاعات برنامه خود را در محلی از هارد کامپیوتر ذخیره کنیم یا این اطلاعات را از هارد کامپیوتر خوانده و دریافت کنیم.

در کل کار با فایل یعنی این که داده های برنامه را در یک فایل در محلی از کامپیوتر ذخیره کنیم و یا این که داده ها را از فایلی درون کامپیوتر بخوانیم.

خب ما در این آموزش میخوایم چکار کنیم؟!

پاسخ: ما در این اموزش قصد داریم روش **ایجاد فایل** ، **ذخیره اطلاعات** درون فایل و **روش خواندن** اطلاعات از فایل را بررسی کنیم.

خب بریم سراغ اصل مطلب!!!

ما برای کار با فایل نیاز به استفاده و import کردن پکیج java.io ابتدای سورس کد برنامه خود داریم.

```
import java.io.*;
```

تقریبا هر کلاسی در جاوا که نیاز به ورودی و خروجی داشته باشد گذرش به استفاده از پکیج java.io خواهد خورد.

Stream

به دنباله ای از داده ها در جاوا یک Stream می گوییم.

حالا این داده ها می توانند از نوع داده های اولیه (متغیرهای اولیه)، اشیا ،کاراکترها و غیره باشند.

ما در جاوا دو نوع Stream داریم:

۱. InPutStream : برای خواندن داده ها از فایل درون کامپیوتر استفاده می شود.
۲. OutPutStream : برای نوشتن و ذخیره داده ها در فایل استفاده می شود.

Byte Streams

byte streams برای انجام ورودی و خروجی های داده های (بایت های) ۸ بیتی استفاده می شود. (خیلی این تعریفا رو جدی نگیرید! فقط میخوام که اینارو گفته باشم!!! اصل مطلب رفتن سراغ مثال عملی هست که جلوتر کار می کنیم پس اصلا نگران نباشید ☺)

اگرچه کلاس های زیادی در زمینه byte streams داریم اما تقریبا بیشتر از FileInputStream و FileOutputStream استفاده می شود.

در زیر مثالی از روش خواندن و نوشتن فایل با استفاده از FileInputStream و FileOutputStream می باشد:

مثال:در مثال زیر قصد داریم یک فایل با فرمت txt از درون کامپیوتر با نام input.txt بخوانیم و اطلاعات درون فایل input.txt را خوانده و در فایل جدید با نام output.txt بریزیم. چیزی شبیه کپی کردن هست یعنی فایل input.txt بخوانیم و هرچی داخلش هست رو بریزیم تو فایل جدیدی با نام output.txt.

- در این مثال فرض بر آن است که از قبل فایل input.txt درون کامپیوتر ما موجود است.

```
package fileIO;

import java.io.*;

public class CopyFile {

 public static void main(String args[]) {
      FileInputStream in = null;
      FileOutputStream out = null;

      try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");

            int c;
            while ((c = in.read()) != -1) {
                  out.write(c);
            }

            in.close();
            out.close();
      } catch (IOException e) {
            System.out.println(e.getMessage());
      }
 }
}
```

خروجی:

```
input.txt (The system cannot find the file specified)
```

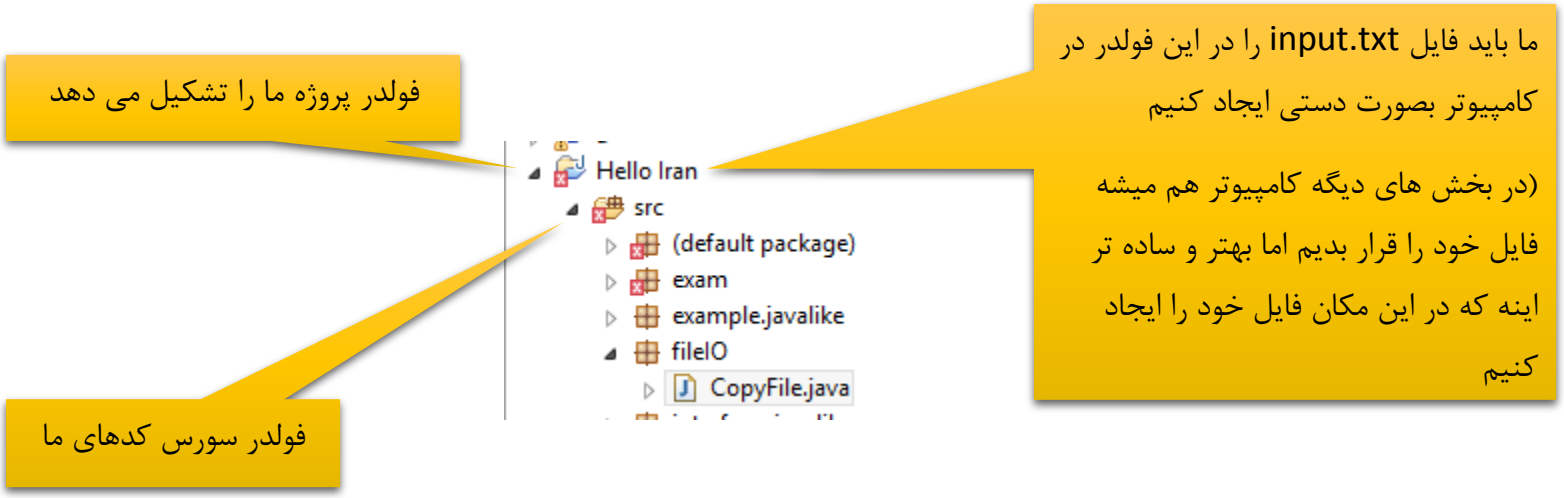
دلیل رخ دادن استثنا در این مثال این هست که هنگام خواندن فایل input.txt ،این فایل در کامپوتر پیدا نشده است!!
همان طور که ابتدای مثال گفتیم ،در این مثال فرض شده که از قبل فایلی با این نام در کامپیوتر ما موجود می باشد.
پس ما یک فایل در مکانی از کامپیوتر خود با نام input و فرمت txt ایجاد کرده و داخلش یک متن می نویسیم!

input.txt

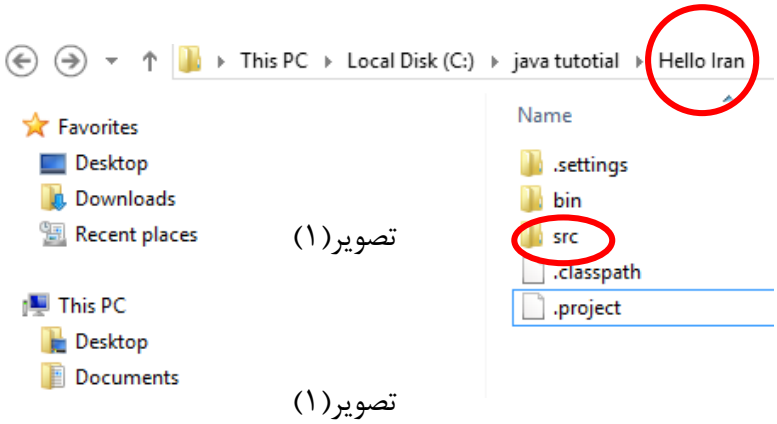
- بخش مشکی نام فایل و بخش قرمز فرمت فایل می باشد.

ما برای ایجاد یک فایل در کامپیوتر بصورت دستی و بدون برنامه نویسی بصورت زیر عمل می کنیم:

۱. ابتدا مکانی از کامپیوتر را برای ایجاد فایل انتخاب می کنیم، بهتر است این مکان داخل **فولدر پروژه** ما در **کنار** فلودر سورس کدهایمان (فولدر src) باشد.

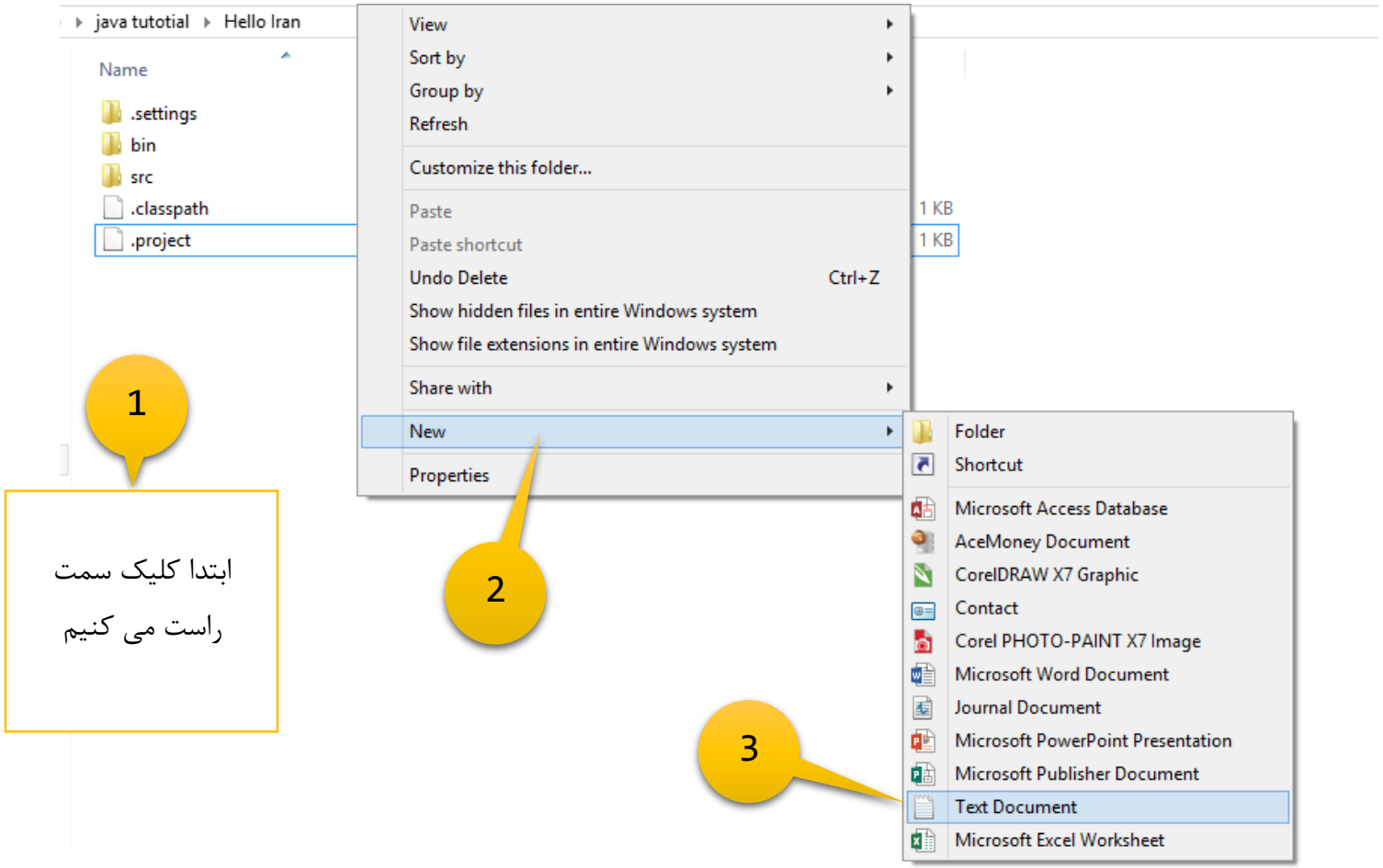


در تصویر(۱) مکان فولدر پروژه Hello Iran را مشاهده می کنید:

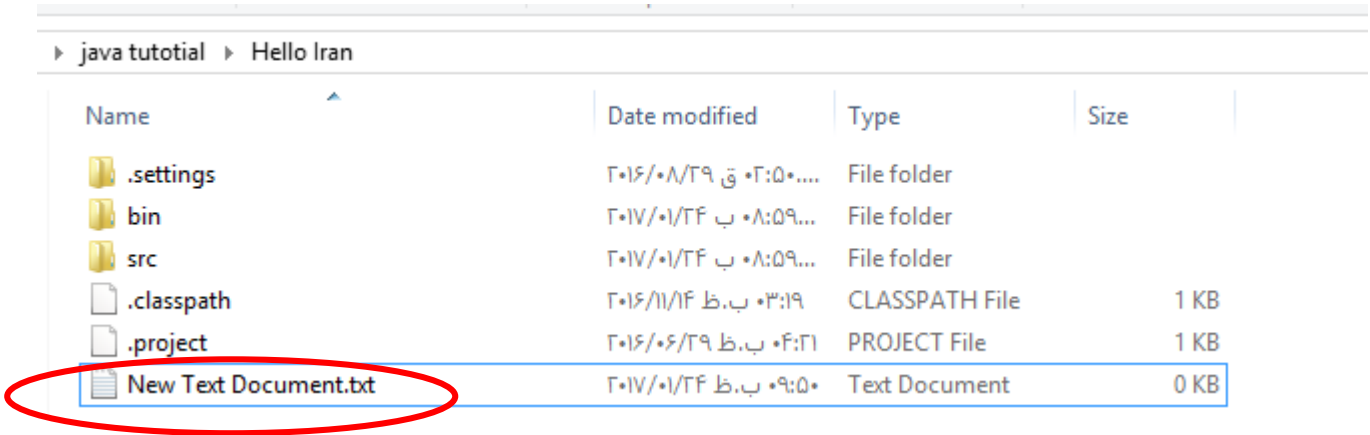


تصویر(۱)

د ر تصویر ۲ و ۳ روش ایجاد یک فایل با فرمت txt و با نام input را در فولدر پروژه مون در کامپیوتر را مشاهده می کنید:

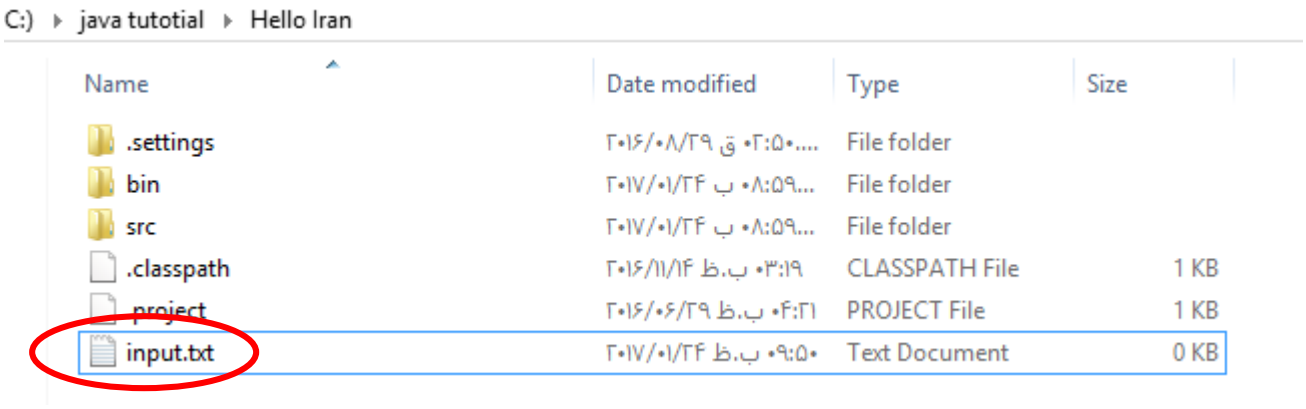


تصویر(۲)

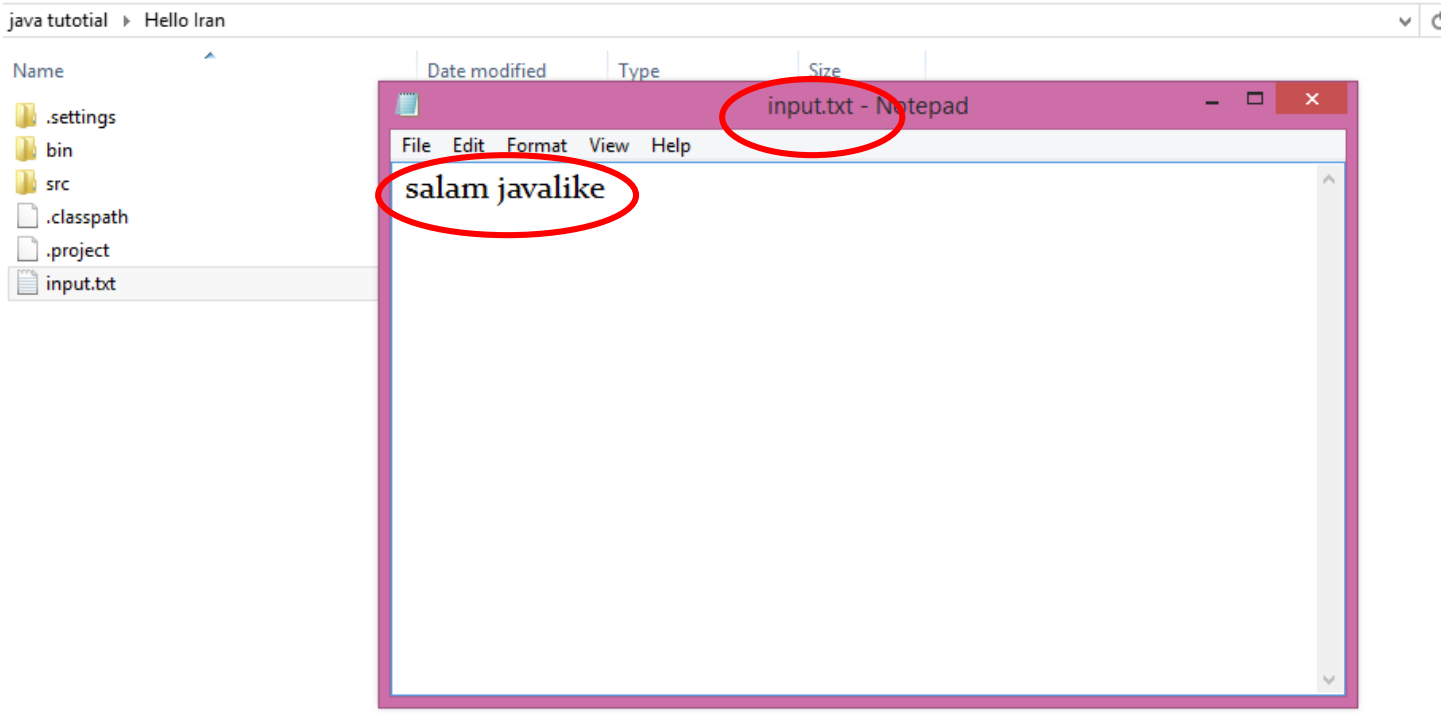


تصویر(۳)

در تصویر ۴ نام فایل New Text Document.txt را به نام input.txt تغییر داده ایم:

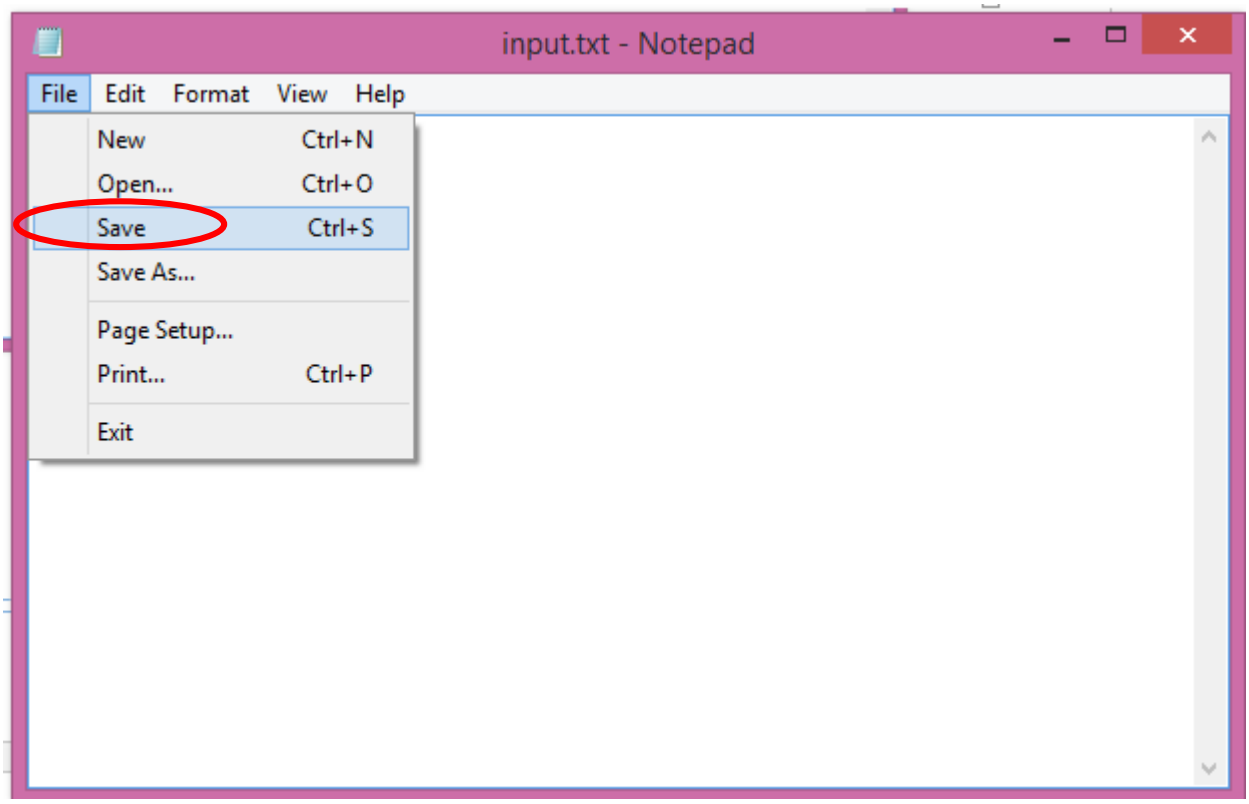


تصویر(۴)



در تصویر ۵ و۶ فایل input.txt باز کرده و متن "salam javalike" را در داخلش می نویسیم و ذخیره(save) می کنیم.

تصویر(۵)



تصویر(۶)

خب فرآیند ایجاد فایل و ریختن داده (در این جا داده ما یک متن بود) بصورت دستی در کامپیوتر تمام شد.

خلاصه کل فرآیند ایجاد فایل در کامپیوتر:

- ایجاد فایل در کامپیوتر به سادگی باز کردن برنامه Notepad در کامپیوتر ونوشتن اطلاعات مورد نظر و ذخیره ان با فرمت txt می باشد، به همین سادگی!!!
- به دلیل این که میخواهیم فایلمون همراه با سورس کدهامون باشه بهتره که فایل مورد نظر در پوشه پروژه هامون در کنار پوشه src قرار گیرد.
- همان طور که میدونید هر فایل که در محیطی از کامپیوتر قرار می گیرد یک آدرس دارد:

مثلا آدرس برنامه آنتی ویروس ند۳۲ در کامپیوتر من بصورت زیر است:

```
C:\Program Files\ESET\ESET Smart Security
```

جالب اینه که برای آدرس فایل های ما در پوشه پروژه مون کافی که تنها نام فایل همراه با فرمت آن را استفاده کنیم:

```
"نام فایل.فرمت"
```

مثلا فایل input.txt آدرسش در پوشه پروژه ما بصورت زیر است:

```
"input.txt"
```

اگه این فایل در درایو D و در پوشه ای با نام files باشد آدرس آن بصورت زیر است:

```
"D:\\files\\input.txt"
```

در جاوا برای آدرس دهی به جای علامت "\" از علامت "\\" استفاده می کنیم یعنی بجای یک بک اسلش از دو بک اسلش استفاده می کنیم.

حالا دیگه خودتون قضاوت کنید کدوم آدرس بهتر است؟!! تازه اگه سورس کد خودتون رو به دوستتون بدید که در کامپیوتر خودش برنامه رو اجرا کنه و دوستتون درایو D نداشت چی؟!! برنامه با خطا روبرو میشه! پس بهتره که فایل های خودمون در پوشه پروژه مورد نظرمون که سورس کد ما درونش قرار داده ،بگذاریم.

خب بریم سراغ کد مثال بالا:

حالا ما تا اینجا فایل input.txt رو در پوشه پروژه مون ایجاد کردن و داخلش متن "salam javalike" نوشتیم.

بار دیگه مثال بالا رو تست می کنیم:

قبل از تست کردن برنامه پوشه پروژه مون رو در تصویر (۵) مشاهده می کنید.

حالا برنامه رو اجرا می کنیم:

```
package fileIO;

import java.io.*;

public class CopyFile {

 public static void main(String args[]) {
     FileInputStream in = null;
     FileOutputStream out = null;

     try {
          in = new FileInputStream("input.txt");
```

```
            out = new FileOutputStream("output.txt");

            int c;
            while ((c = in.read()) != -1) {
                  out.write(c);
            }

            in.close();
            out.close();
      } catch (IOException e) {
            System.out.println(e.getMessage());
      }
 }
}
```

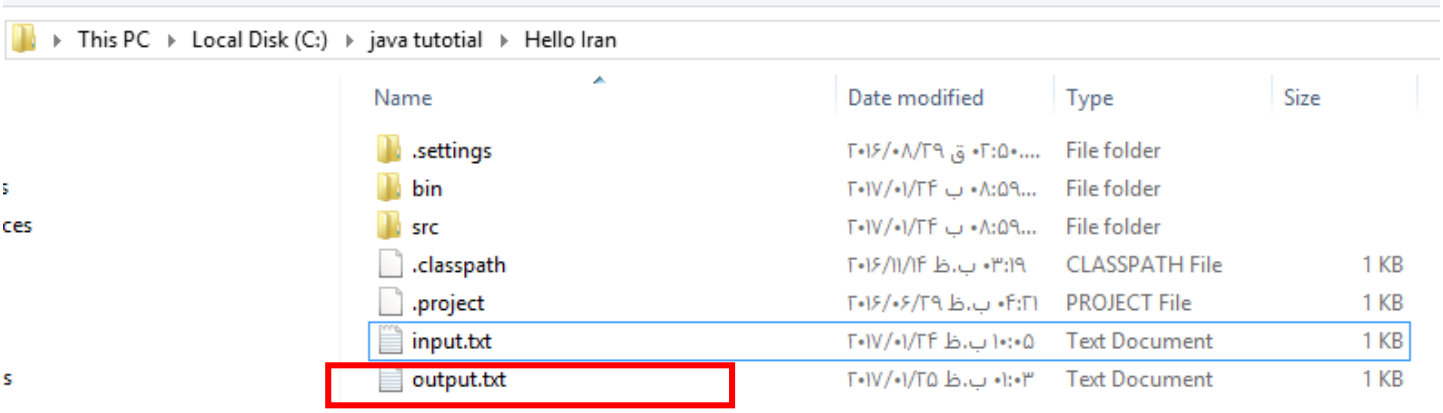
خروجی:

```

```

در محیط کنسول چیزی نمایش داده نمی شود! اما این کد برنامه هدفش این بود که اطلاعات فایل input.txt رو بخواند و درون یک فایل جدید با نام output.txt بریزد.(کپی کردن فایل input.txt درون فایل output.txt)

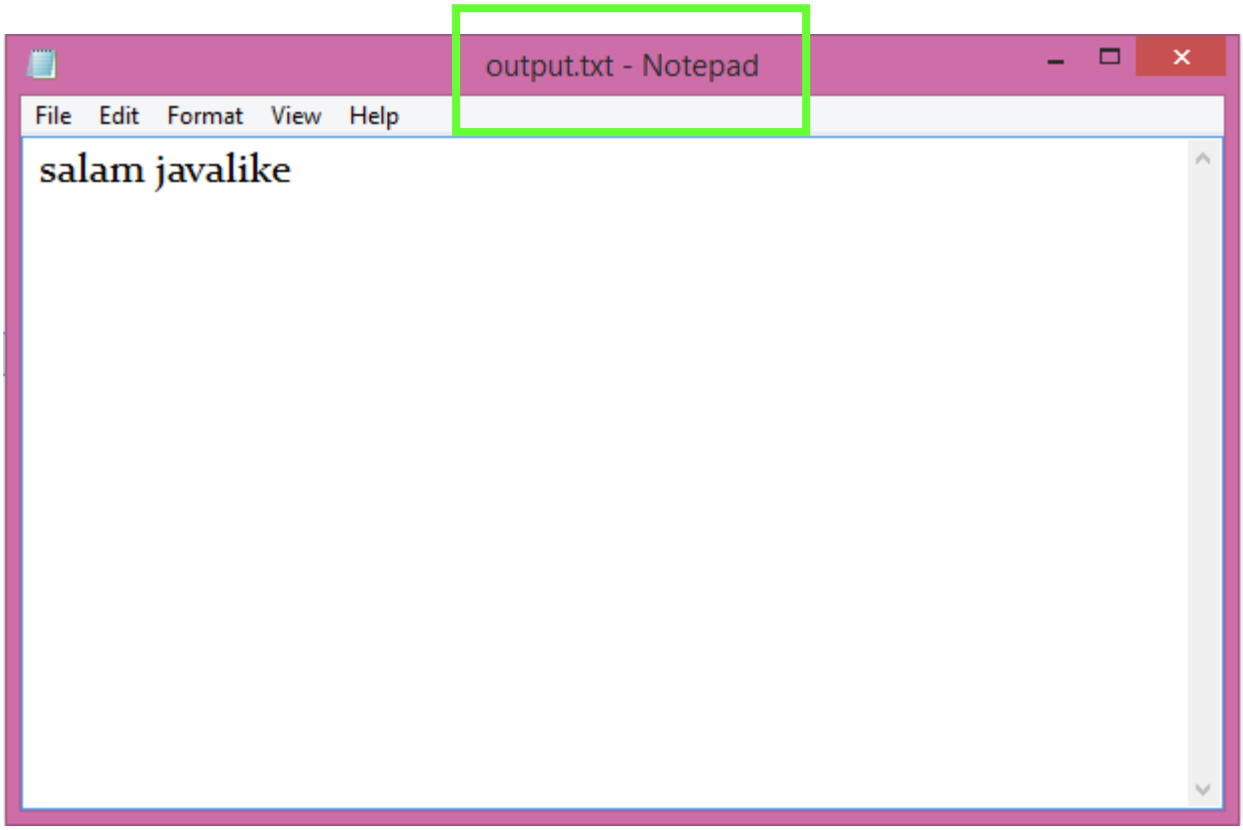
پس یک فایل جدید با نام output.txt که حاوی اطلاعات کپی شده از فایل input.txt است ایجاد می شود.

برای این که ببینیم واقعا فایل input.txt کپی شده و فایل output.txt ایجاد شده یا خیر به سراغ پوشه پروژه مون می رویم:تصویر(۷)



تصویر(۷)

همان طور که در تصویر ۷ می بینید فایل output.txt ایجاد شده است.حالا در تصویر(۸) این فایل را باز می کنیم:



تصویر(۸)

همان طور که در تصویر ۸ مشاهده می کنید اطلاعات فایل `input.txt` درون فایل `output.txt` کپی شده است.

عملکرد این برنامه:

این برنامه یک فایل با نام `input.txt` را از کامپیوتر پوشه پروژمون خواند و یک فایل جدید با نام `output.txt` ایجاد کرد و اطلاعات فایل `input.txt` را درون فایل `output.txt` ریخت.

حالا بررسی سورس کد برنامه :

```
package fileIO;

import java.io.*;

public class CopyFile {

 public static void main(String args[]) {
     FileInputStream in = null;
     FileOutputStream out = null;

     try {
          in = new FileInputStream("input.txt");
```

```
            out = new FileOutputStream("output.txt");

            int c;
            while ((c = in.read()) != -1) {
                  out.write(c);
            }

            in.close();
            out.close();
      } catch (IOException e) {
            System.out.println(e.getMessage());
      }
 }
}
```

```
package fileIO;
```

نام پکیج ما که سورس کد برنامه ما داخلش قرار دارد.

```
import java.io.*;
```

چون قصد داریم با فایل در جاوا کار کنیم پکیج java.io.* را import می کنیم.

```
public class CopyFile {

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

کلاس و متد main برنامه را مشاهده می کنید..

```
      FileInputStream in = null;
      FileOutputStream out = null;
```

ما برای ورودی و خروجی گرفتن فایل نیاز داریم از کلاس های FileInputStream و FileOutputStream شی ایجاد کنیم. کلاس های متفاوتی برای ورودی و خروجی گرفتن از فایل وجود دارد که موارد مذکور یکی از این کلاس هاست.

- ورودی یعنی همان خواندن داده ها از فایل
- خروجی یعنی همان ایجاد و نوشتن (ریختن) داده ها درون فایل

```
      try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");
```

```
        int c;
        while ((c = in.read()) != -1) {
            out.write(c);
        }

        in.close();
        out.close();
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
```

همان طور که در مبحث استثناها بررسی کردیم بعضی مواقع در برنامه دچار استثناهایی در زمان کامپایل و اجرای برنامه می شویم.

در مبحث کار با فایل نیز ما در زمان کامپایل دچار استثنا می شویم.زیرا امکان داره هنگام خواندن فایل از کامپیوتر ،فایل مورد نظر وجود نداشته باشه و.... پس ما برای پیش بینی همچین موردی با دستور `try-catch` استثنایی که برای فایل هایمون رخ میدهد را کنترل می کنیم.

```
in = new FileInputStream("input.txt");
```

در اینجا سازنده شی `in` که از نوع کلاس `FileInputStream` می باشد را صدا زده ایم.

بجای پارامتر درون پرانتز جلوی سازنده که از نوع `String` می باشد، آدرس فایل مورد نظرمون را قرار می دهیم.چون فایل ما در پوشه پروژه مون قرار داره آدرس بصورت زیر داده ایم:

`"input.txt"`

```
out = new FileOutputStream("output.txt");
```

در اینجا سازنده شی `out` که از نوع کلاس `FileOutputStream` می باشد را صدا زده ایم.

بجای پارامتر درون پرانتز جلوی سازنده که از نوع `String` می باشد، آدرس مکانی که این فایل بعد از ایجاد در آن قرار می گیرد را قرار می دهیم.

کلاس `FileInputStream` وقتی شی ازش ایجاد میکنیم و به سازنده ان ادرس میدیم ، در سیستم به محل آدرس داده شده می رود، و به دنبال فایل با آدرس مورد نظر میگردد و درصورت پیدا شدن فایل آن را باز می کند.

کلاس `FileOutputStream` وقتی شی ازش ایجاد میکنیم و به سازنده آن ادرس میدیم یک فایل برامون ایجاد می کند.

```
int c;
while ((c = in.read()) != -1) {
out.write(c);
}
```

این چند خط کد بالا برای خواندن اطلاعات فایل باز شده و نوشتن و ذخیره (ریختن) این اطلاعات در فایل خروجی ما می باشد.

در اینجا فایل ورودی و در حال خوانده شدن ما **in** نام دارد و و فایل جدید و خروجی ما **out** نام دارد.

در اینجا یک متغیر به نام **c** از نوع **int** تعریف کرده ایم.

در حلقه **while** دستور **(c = in.read()) != -1** را مشاهده می کنید.در اینجا کاراکتر به کاراکتر اطلاعات درون فایل خوانده می شود و معادل عددی هر کاراکتر را برمی گرداند و درون متغیر **c** ریخته می شود و تا هنگامی که مقدار متغیر **c** برابر **-1** نشده حلقه ادامه دارد. و با هربار خوانده شدن ،کاراکتر از طریق متد **write** درون فایل **out** ریخته می شود.

```
in.close();
out.close();
```

در پایان هر دو فایل ورودی و خروجی با دستور **close()** می بندیم.

```
 } catch (IOException e) {
          System.out.println(e.getMessage());
     }
```

در صورت رخ دادن استثنا مربوط به فایل متد **catch** صدا زده می شود و دستورات بلوک درون آن اجرا می شود.

خب تا اینجا روش خواندن و نوشتن فایل به کمک کلاس های **FileInputStream** و **FileOutputStream** را یادگرفتیم.

حالا قصد داریم مثال بالا را بصورت زیر تغییر دهیم:

در مثال زیر ابتدا قصد داریم که از کاربر درخواست بشه که یک متن به ورودی دهد و یک فایل جدید در پوشه پروژه مون ایجاد بشه و متن داده شده را درون فایل ریخته و ذخیره کنیم:

```
package fileIO;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Scanner;

public class Createfile {
```

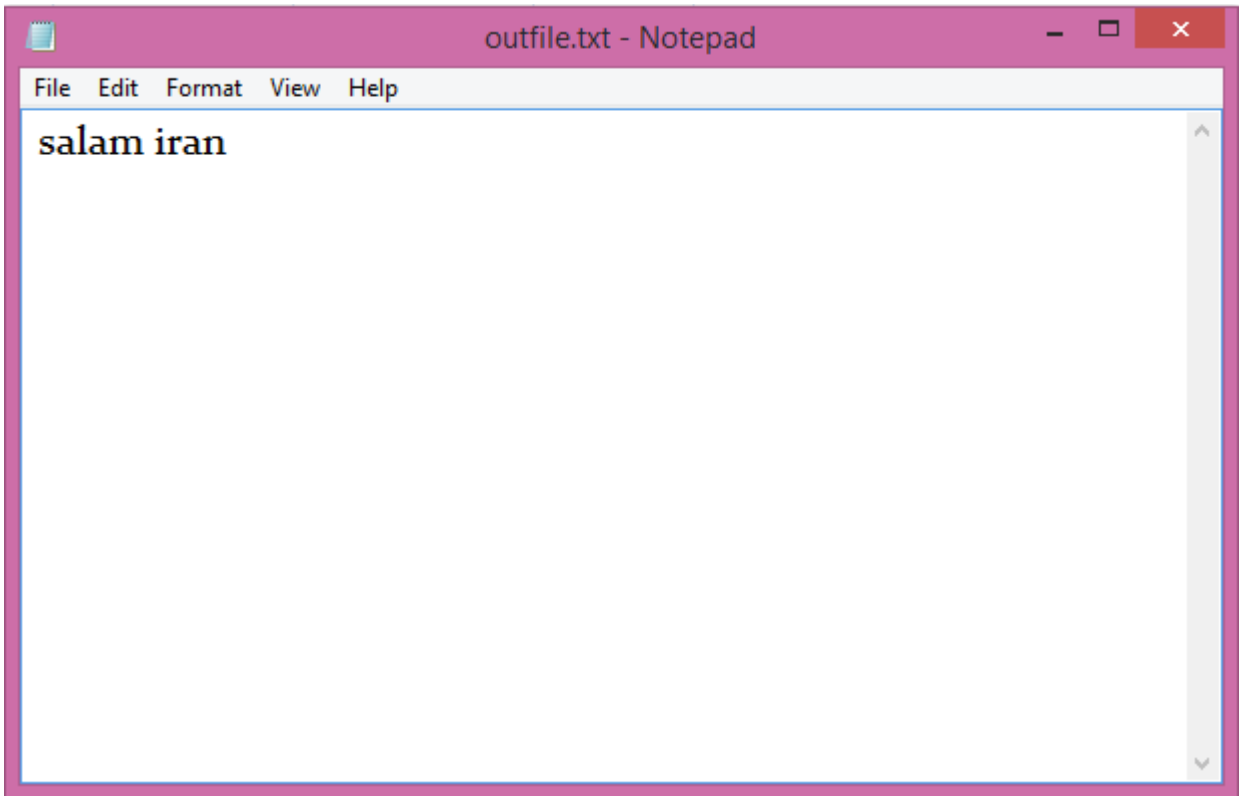
```
 public static void main(String args[]) {
      FileOutputStream out = null;
      Scanner input = new Scanner(System.in);
      System.out.println("Please Enter the Text:");
      String text = input.nextLine();
      byte tx[] = new byte[text.length()];
      for (int i = 0; i < text.length(); i++) {

            tx[i] = (byte) text.charAt(i);
      }

      try {

            out = new FileOutputStream("output.txt");

            out.write(tx);

            out.close();
      } catch (IOException e) {
            System.out.println(e.getMessage());
      }
 }
}
```

خروجی:

```
Please Enter the Text:
salam iran
```

بر فرض متن **"salam iran"** را از ورودی کیبرد گرفته ایم.

در پوشه پروژه مون فایلی با نام و فرمت **outfile.txt** ایجاد می شود. که محتوای درون فایل ما بصورت تصویر ۹ می باشد



تصویر(۹)

پس در این برنامه ما توانستیم متنی از ورودی کیبرد بگیریم و یک فایل ایجاد کنیم و درون فایل متن دریافت شده رو بریزیم.

```
Scanner input = new Scanner(System.in);
```

برای ورودی گرفتن از کیبرد از کلاس Scanner شی ساخته ایم.

```
System.out.println("Please Enter the Text:");
     String text = input.nextLine();
```

رشته یا متن دریافت شده رو درون یک متغیر از نوع String با نام text میریزیم.

```
byte tx[] = new byte[text.length()];
     for (int i = 0; i < text.length(); i++) {

          tx[i] = (byte) text.charAt(i);
     }
```

چرا یک آرایه از نوع byte تعریف کردیم؟! پاسخ : ابتدای آموزش گفتیم که کلاس FileOutputStream بصورت بایت بایت اطلاعات و داده ها را درون فایل ذخیره می کند ،پس اینجا هم مجبوریم که تک به تک کاراکتر های متن دریافت شده از ورودی کیبرد را به بایت تبدیل کرده و درون یک ارایه از نوع بایت بریزیم.

طول آرایه برابر است با تعداد کاراکتر متن دریافت شده از ورودی می باشد.

بعد با استفاده از یک حلقه تک به تک کاراکترهای متغیر **text** را استخراج کرده و با استفاده از دستور (**byte**) کاراکترها را به معادل عددی ان از نوع بایت تبدیل ساخته ایم به این عمل **casting** در جاوا می گویند که سر فرصت بهش می پردازیم و اینجا اگه بهش بپردازیم اموزش طولانی و به حاشیه می رویم.در کل فعلا فراموشش کنید!!

```
out = new FileOutputStream("output.txt");
```

در اینجا یک فایل با نام **output** و با فرمت **txt** ایجاد کرده ایم.

```
out.write(tx);
```

در اینجا یکجا آرایه **tx** که از نوع بایت هستش و هر خانه آن حاوی مقدار عددی کاراکترهای متن ما می باشد را داخل فایل میریزیم. خب اینجا به بعدش هم دیگه متد **write** خودش کارشو بلده و قشنگ تک تک کاراکترها رو کنار هم قرار میده و متن ما را در فایل تشکیل می دهد .

```
    out.close();
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
}
```

در نهایت فایل **out** را می بندیم.

متد **catch** هم در صورت رخ دادن استثنا صدا زده می شود.

ما اول آموزش بصورت دستی در ویندوز یک فایل تکس در **notepad** ایجاد کردیم و داخلش اطلاعات مورد نظر را ریختیم و ذخیره کردیم. حالا ما بصورت کد برنامه نویسی یک فایل ایجاد کردیم و داخلش اطلاعات مورد نظرمون رو ریختیم.

- کلاس ها و روش های دیگر خواندن و نوشتن فایل را در ادامه بررسی می کنیم.

Character Streams

در جریان **Byte streams** (دنباله ای از داده های از نوع بایت) که بررسی کردیم،فایل های ما بایت های (داده های) ۸ بیتی را در ورودی می خواندن یا در خروجی می نوشتند.

Character Streams (دنباله ای از داده های کاراکتری) برای انجام ورودی و خروجی های یونیکد ۱۶ بیتی استفاده می شود.

کلاس های زیادی برای خواندن و نوشتن فایل در زمینه Character Streams هستند که ما دو کلاس پرکاربرد آن یعنی FileReader و FileWriter بررسی می کنیم.

```
package fileIO;

import java.io.*;

public class CopyFile {

	public static void main(String args[]) {

		try {
			FileReader in= new FileReader("input.txt");
			FileWriter out = new FileWriter("output.txt");

			int c;
			while ((c = in.read()) != -1) {

				out.write(c);
			}
			in.close();
			out.close();

		} catch (IOException e) {
			System.out.println(e.getMessage());
		}
	}
}
```

خروجی:

```

```

در کنسول خروجی نداریم اما در پوشه پروژمون یک فایل جدید با نام output.txt ایجاد می شود که حاوی داده هایی است که از فایل input.txt کپی شده است.

```
FileReader in= new FileReader("input.txt");
```

برای خواندن فایل موجود در آدرس "input.txt" از کلاس FileReader شی ساخته ایم.

```
FileWriter out = new FileWriter("output.txt");
```

یک شی با نام out از کلاس FileWriter تعریف کرده ایم.با این کار یک فایل جدید با نام output در پوشه پروژه ما ایجاد می شود.

```
    int c;
            while ((c = in.read()) != -1) {

                out.write(c);
            }
```

بایت به بایت فایل in خوانده می شود و درون فایل out ریخته می شود تا زمانی که به بایت -۱ برسیم و فرآیند خواندن فایل به اتمام میرسد و از حلقه خارج می شویم.

```
    in.close();
            out.close();

        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
```

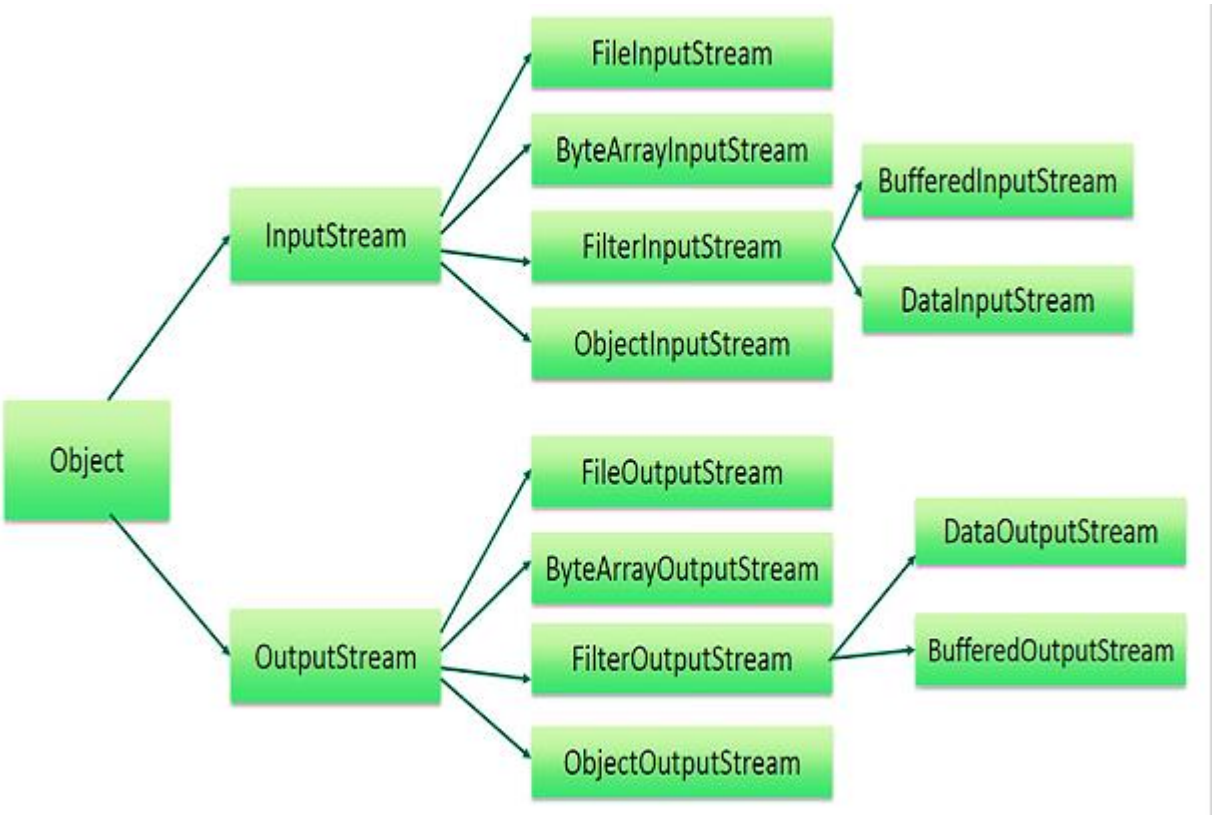
بعد از اتمام کار با فایل ،دو فایل in و out را با متد close می بندیم.

در صورت رخ دادن استثنای احتمالی دستور درون متد catch اجرا می شود.

خواندن و نوشتن فایل ها(Reading and Writing Files):

همانطور که قبلا توضیح دادیم،یک stream دنباله ای از داده ها می باشد.ما می توانیم از کلاس های InputStream برای خواندن داده ها و از OutputStream برای نوشتن و ذخیره داده ها در مقصد مورد نظر استفاده کنیم.

در تصویر ۱۰ سلسه مراتب ورودی (خواندن) و خروجی (نوشتن) جریان داده ها در فایل را مشاهده می کنید:



تصویر(۱۰)

در این آموزش به دو جریان مهم خواندن و نوشتن فایل در جاوا یعنی `FileInputStream` و `FileOutputStream` می پردازیم.

:FileInputStream

این کلاس برای خواندن داده ها از فایل استفاده می شود. می توانیم از با استفاده از کلمه کلیدی **new** و صدا زدن یکی از سازنده های این کلاس (این کلاس چندین سازنده دارد) از کلاس `FileInputStream` شی ایجاد کنیم.

```
FileInputStream in = new FileInputStream("input.txt");
```

در بالا به سازنده کلاس یک رشته به عنوان آدرس محل فایل در کامپیوتر را داده ایم.ما میتونیم ابتدا یک متغیر از نوع `String` تعریف کنیم و آدرس را درونش بریزیم و در نهایت نام متغیر را داخل سازنده کلاس قرار بدیم:

```
String address="input.txt";
FileInputStream in = new FileInputStream(address);
```

همچنین می توانیم ابتدا از کلاس File یک شی ایجاد کنیم و درون سازنده آن آدرس مکان فایل مورد نظر در کامپیوتر را قرار بدیم و بعدش شی ساخته شده از نوع کلاس File را درون سازنده کلاس FileInputStream بریزیم.

```
File f=new File("input.txt");
FileInputStream in = new FileInputStream(f);
```

طبق نمودار تصویر ۱۰ کلاس FileInputStream فرزند کلاس InputStream می باشد،پس طبق مبحث چندریختی که در جلسات قبل خواندیم می توانیم شی ای از نوع کلاس InputStream ایجاد کنیم که سازنده آن ،از سازنده کلاس فرزند InputStream یعنی FileInputStream می باشد.

```
File f=new File("input.txt");
InputStream in = new FileInputStream(f);
```

همان طور که مشاهده می کنید شی in از نوع InputStream و سازنده آن از نوع کلاس FileInputStream می باشد، طبق مبحث چندریختی آزاد هستیم که یک شی را از نوع کلاس پدر و سازنده شی را از نوع فرزند کلاس پدر انتخاب کنیم.

دلیل این که ما دست به چنین اقدامی زدیم این هست که به متدهای کلاس InputStream دست پیدا کنیم.

- در زیر لیستی از متدهای کلاس InputStream را مشاهده می کنید که از طریق آنها می توانیم فایل خود را دستکاری و روی فایلمون عملیات خاص خود را انجام بدیم:

1.public void close() throws IOException{}

- بعد از این که کارمون با فایل تمام شد با این متد فایل خود را می بندیم.
- برای کنترل استثنا احتمالی این متد throws شده به استثنای IOException

2.public int read()throws IOException{}

- این متد برای خواندن بایت به بایت داده های درون فایل می باشد.این متد داده های درون فایل را بصورت عدد صحیح از نوع byte برمی گرداند. مثلا اگر داخل فایلمون حرف a وجود داشته باشد ،این حرف را بصورت عدد معادل یعنی ۹۷ برمی گرداند.(همان طور که میدانید شماره کاراکتر a برابر ۹۷ می باشد)
- این متد همین جور بایت به بایت داده های درون فایل رو میخونه و جلو میره تا زمانی که به عدد -۱ برخورد کند که باعث تمام شدن عملیات خواندن داده های درون فایل می شود.

3.public int available() throws IOException{}

این متد تعداد بایت های موجود در فایل را به ما می دهد.منظور از تعداد بایت همان تعداد معادل عددی کاراکترهای موجود در فایل می باشد.

FileOutputStream

کلاس FileOutputStream برای ایجاد و نوشتن داده ها در آن استفاده می شود.یعنی ابتدا یک فایل جدید در مکان مورد نظر کامپیوتر ایجاد میکنه بعدش با متد مربوطه داده هامون رو داخلش میریزیم.
برخلاف کلاس FileInputStream که باید از قبل فایل در کامپیوتر موجود باشد،کلاس FileOutputStream فایلی را ایجاد میکند که از قبل در کامپیوتر موجود نیستش.

هنگام شی سازی از کلاس FileOutputStream از یکی از دو سازنده پرکاربرد آن می توانیم استفاده کنیم:
یکی سازنده ای که یک String به عنوان آدرس محل ایجاد فایل دریافت می کند:

```
FileOutputStream out=new FileOutputStream(“out.txt”);
```

و دیگری سازنده ای که شی ای از نوع کلاس File به عنوان پارامتر دریافت می کند:

```
File f=new File("out.txt");
FileOutputStream out=new FileOutputStream(f);
```

طبق تصویر ۱۰ کلاس FileOutputStream فرزند کلاس OutputStream می باشد.پس به روش چندریختی به شکل زیر می توان از این کلاس شی ساخت:

```
OutputStream out=new FileOutputStream("out.txt");
```

در اینجا می توانیم از هر یک از دو سازنده این کلاس استفاده کرد.
وقتی نوع شی ما از نوع کلاسOutputStream انتخاب شود به لیستی از متدهای این کلاس دسرسی پیدا می کنیم که در زیر برخی از آنها را بررسی کرده ایم:

public void close() throws IOException{}

این متد برای بستن فایل ایجاد شده استفاده می شود.

public void write(int w)throws IOException{}

این متد برای نوشتن و ریختن داده هامون داخل فایل استفاده می شود. ورودی این متد یک عدد صحیح از نوع int هستش ،معنای آن این است که داده های ما برای نوشتن در فایل باید بصورت بایت بایت باشد،مثلا اگه خواستیم کاراکتر a را درون یک فایل بریزیم با استفاده از این متد ، بجای ریختن مستقیم کاراکتر a معادلی عددی (بایتی) این حرف یعنی 97 را درون متد write قرار می دهیم.

public void write(byte[] w)

این متد هم آرایه از داده ها از نوع بایت را درون فایل می ریزد.

خب حالا بریم سراغ مثال که هیچ توضیحی به اندازه مثال کارساز نیست ☺ اگه از توضیحات چیزی دستگیرتون نشد نگران نباشید! مثال حل کنید!!جواب میگیرید.

مثال) در زیر کاربرد کلاس های InputStream و OutputStream را خواهیم دید:

```
package fileIO;
import java.io.*;
public class fileStreamTest {

   public static void main(String args[]) {

      try {
         byte bWrite [] = {60,70,80,120,61};
         OutputStream os = new FileOutputStream("test.txt");
         for(int x = 0; x < bWrite.length ; x++) {
            os.write( bWrite[x] );   // writes the bytes
         }
         os.close();

         InputStream is = new FileInputStream("test.txt");
         int size = is.available();

         for(int i = 0; i < size; i++) {
            System.out.print((char)is.read() + "  ");
         }
         is.close();
      }catch(IOException e) {
         System.out.print("Exception");
      }
   }
}
```

خروجی:

```
< F P x =
```

```
try {
```

دلیل استفاده از بلوک try-catch کنترل رخ دادن استثناهای احتمالی مربوط به خواندن و نوشتن فایل می باشد.

```
byte bWrite [] = {60,70,80,120,61};
```

آرایه ای از نوع byte تعریف و مقدار دهی اولیه شده است.

```
OutputStream os = new FileOutputStream("test.txt");
```

یک شی از نوع کلاس OutputStream به روش چندریختی ایجاد کرده ایم و به ورودی پارامتر سازنده اش یک آدرس برای محل ایجاد فایل داده ایم.

```
        for(int x = 0; x < bWrite.length ; x++) {
           os.write( bWrite[x] );   // writes the bytes
        }
```

این حلقه خانه های آرایه که حاوی تعدادی مقادیر از نوع بایت هستند را پیمایش می کند، و هرخانه از آرایه که بایت های آن معادل یک کاراکتر حرفی هست را داخل فایل می نویسد.

```
os.close();
```

بعد از تمام شدن کارمون با فایل os آن را می بندیم.

```
InputStream is = new FileInputStream("test.txt");
```

یک شی برای خواندن فایل از نوع InputStream به روش چندریختی ایجاد کرده ایم.ورودی سازنده آدرس محل فایلی است که قراره آن را بخوانیم.

```
int size = is.available();
```

این دستور تعداد بایت (کاراکتر) های موجود در فایل را برمی گرداند و میریزه داخل متغیر size

```
for(int i = 0; i < size; i++) {
System.out.print((char)is.read() + "  ");
}
```

به تعداد بایت ها (کاراکترها) موجود در فایل ،متد read را برای خواندن بایت به بایت فایل صدا می زنیم.با هر بار صدا زدن متد read یک بایت یا معادل عددی یک کاراکتر به ما برمی گرداند.

```
           is.close();
        }catch(IOException e) {
           System.out.print("Exception");
        }
```

در پایان کار با فایل is با متد close آن را می بندیم.

متد catch هم اگه استثنایی در خواندن یا نوشتن فایل رخ دهد دستورات درون آن اجرا می شود.

- مبحث فایلها در جاوا بسیار بزرگ است که در این آموزش اشاره ای به آن کردیم در آینده تیکه تیکه آموزش درمورد این مبحث طراحی خواهیم کرد که هم سنگین نباشه و هم راحت تر یاد بگیریم

پیروز و موفق باشید

سایت آموزش زبان جاوا به زبان ساده،آسان و شیرین!!!

www.JAVAPro.ir

آموزش جاوا SE را با تجربه شخصی و به زبان خودمونی یاد بگیرید!!!!!

بازدید ازکانال

بازدید از سایت

هر روز مفاهیم و مثال های جدید به سایت اضافه می شود برای اطلاع از مطالب جدید روی سایت عضو کانال شوید.