دانشکده ریاضی و علوم کامپیوتر

برنامه‌سازی پیشرفته و کارگاه

کار با فایل

استاد درس

دکتر مهدی قطعی

استاد دوم

بهنام یوسفی مهر

نگارش

سیدآرمان حسینی، مهدی جعفری

بهار ۱۴۰۳

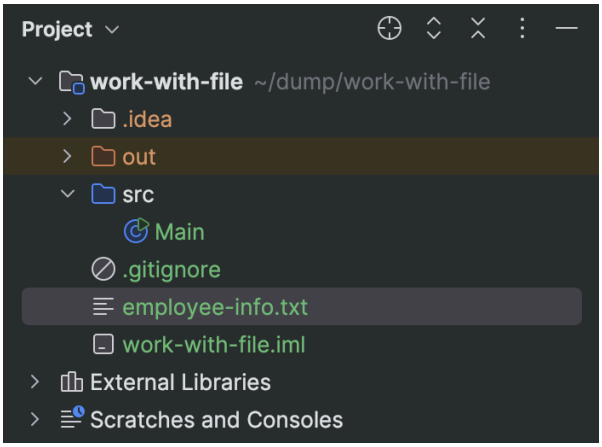
فهرست

خواندن از فایل

کلاس Scanner با File

تا این‌جای کار، شما با کلاس File و نحوهٔ کار با اون آشنا شدید. حالا می‌خوایم یاد بگیریم که چطور می‌شه محتوای یک فایل رو خوند و به اون چیزهای جدید اضافه کرد.

یه پروژهٔ جدید توی IntelliJ ایجاد کنید. توی دایرکتوری خود پروژه (خارج از دایرکتوری src)، فایل employee-info.txt رو ایجاد کنید:



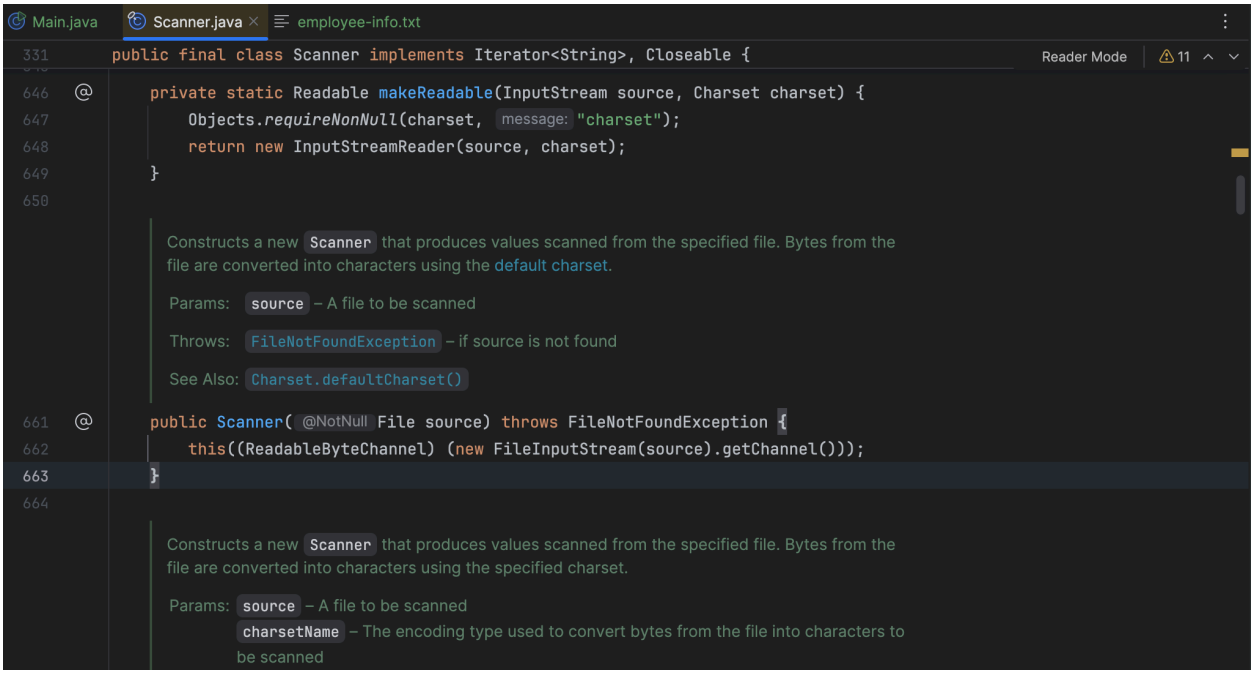
توی این فایل اطلاعات زیر رو بنویسید:

```
Name: Raees - JobTitle: Riasat
Name: Gholi - JobTitle: Developer
Name: Mamad - JobTitle: Developer
```

سیوش کنید و به Main برید. حالا یک آبجکت فایل برای employee-info.txt درست کنید:

```
import java.io.File;

public class Main {
    public static void main(String[] args) {
        var file = new File("employee-info.txt");
    }
}
```

حالا برای خوندن این فایل، از Scanner استفاده می‌کنیم. همون Scannerای که برای ورودی گرفتن از کاربر هم ازش استفاده می‌کردیم. این کلاس، یک کانستراکتورِ overload شده داره که File ورودی می‌گیره:



همون‌طور که می‌بینید، ممکنه که این کانستراکتور FileNotFoundException بده، با توجه به این که این اکسپشن checkedئه، حین استفاده از این کانستراکتور باید حتما اون رو catch کنید (یا به method signitureتون اضافه‌ش کنید). یک آبجکت Scanner برای فایل‌مون می‌سازیم:

```
Scanner scn;

try {
    scn = new Scanner(file);
} catch (FileNotFoundException e) {
    System.err.println("file \"" + file.getAbsolutePath() + "\" does not
exist");
    return;
}
```

و سپس، مثل وقتی که از کاربر ورودی می‌گرفتیم شروع به خوندن فایل می‌کنیم و اون رو به کاربر خروجی می‌دیم:

```
while (scn.hasNextLine()) {
    String employee = scn.nextLine();
    System.out.println("Employee info: \n\t" + employee);
}
```

اگر این کد رو اجرا کنید، خروجی زیر رو می‌بینید:

```
Employee info:
    Name: Raees - JobTitle: Riasat
Employee info:
    Name: Gholi - JobTitle: Developer
```

```
Employee info:
    Name: Mamad - JobTitle: Developer
```

دقت کنید که ما، از یک متد جدید کلاس Scanner به اسم hasNextLine هم توی این کد استفاده کردیم. این متد، زمانی که خط جدیدی از فایل مونده باشه true و در غیر این صورت false خروجی می‌ده.

هر کاری که قبلا با Scanner می‌کردین، این‌جا هم می‌تونید بکنید. مثلا اگر فایلی از یک سری عدد تشکیل شده باشه می‌تونید با استفاده از متدهای ()nextInt و ()hasNextInt اون‌ها رو بخونین:

```
while (scn.hasNextInt()) {
    int num = scn.nextInt();
    System.out.println("Number: " + num);
}
```

اگر خوندن فایلی تموم شده باشه و scannerتون چیز جدیدی برای خوندن نداشته باشه، متدهایی مثل ()nextLine و ()nextInt اکسپشنی از جنس NoSuchElementException می‌دن.

کلاس Scanner با String

کلاس Scanner یک کانستراکتور هم داره که ازتون String ورودی می‌گیره:

```
Scanner scn = new Scanner("10 20 30 40 50");
```

توی این کد، Scanner به جای خوندن از فایل یا ورودی کاربر، رشتهٔ ورودی کانستراکتورش رو می‌خونه. مثلا می‌تونید اعداد رشتهٔ بالا رو به شکل زیر بخونید:

```
while (scn.hasNextInt()) {
    int num = scn.nextInt();
    System.out.println("Num: " + num);
}
```

خروجی این کد به شکل زیره:

```
Num: 10
Num: 20
Num: 30
Num: 40
Num: 50
```

برای این که از این کانستراکتور کلاس Scanner بهتر استفاده کنیم، فایل employee-info.txt رو به شکل زیر تغییر بدید:

```
Raees 09123456789 50
Gholi 09028789123 25
Mamad 09361889898 30
```

توی هر خط این فایل جدید، اطلاعات یک کارمند اومده. اولین بخش هر خط اسم اون کارمند، دومین بخشش شماره تماسش و بخش سوم اون، سنشه. ما می‌خوایم با خوندن اطلاعات هر کارمند از این فایل اون رو پرینت کنیم.

برای این کار، ابتدا با استفاده از کد زیر خط به خط فایل رو می‌خونیم و اون رو به متد printEmployeeInfo ورودی می‌دیم:

```
public static void main(String[] args) {
    var file = new File("employee-info.txt");

    Scanner scn;

    try {
        scn = new Scanner(file);
    } catch (FileNotFoundException e) {
        System.err.println("file \"" + file.getAbsolutePath() + "\" does not
exist");
        return;
    }

    while (scn.hasNextLine()) {
        System.out.println("Employee: ");
        printEmployeeInfo(scn.nextLine());
    }
}
```

و توی متد printEmployeeInfo با استفاده از یه Scanner جدید شروع به خوندن ورودی می‌کنیم و اطلاعات هر کارمند رو خروجی می‌دیم:

```
private static void printEmployeeInfo(String info) {
    Scanner scn = new Scanner(info);

    String name = scn.next();
    String phoneNumber = scn.next();
    int age = scn.nextInt();

    System.out.println("\tName: " + name);
    System.out.println("\tPhone: " + phoneNumber);
    System.out.println("\tAge: " + age);
}
```

خروجی این کد به شکل زیره:

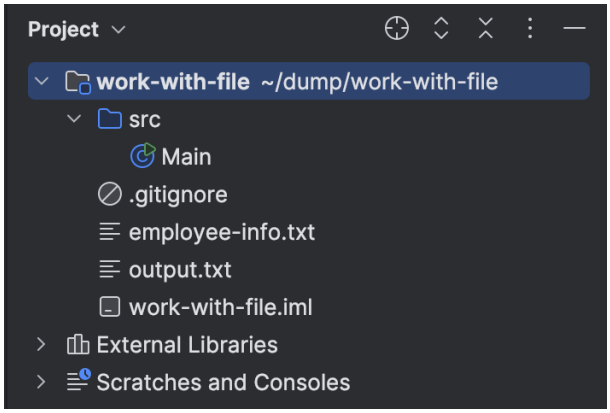
```
Employee:
    Name: Raees
    Phone: 09123456789
```

```
    Age: 50
Employee:
    Name: Gholi
    Phone: 09028789123
    Age: 25
Employee:
    Name: Mamad
    Phone: 09361889898
    Age: 30
```

نوشتن به فایل

کلاس PrintStream

برای نوشتن به فایل‌ها، از کلاس PrintStream استفاده می‌کنیم. برای این که یه مقدار با این کلاس کار کنیم، اول فایل output.txt رو توی دایرکتوری پروژه‌تون ایجاد کنید:



و بعد، توی کدتون یه آبجکت File برای این فایل درست کنید:

```
public static void main(String[] args)  {
    var file = new File("output.txt");
}
```

حالا، شبیه شکلی که قبلا Scanner درست می‌کردین، یه آبجکت PrintStream برای این فایل درست کنید:

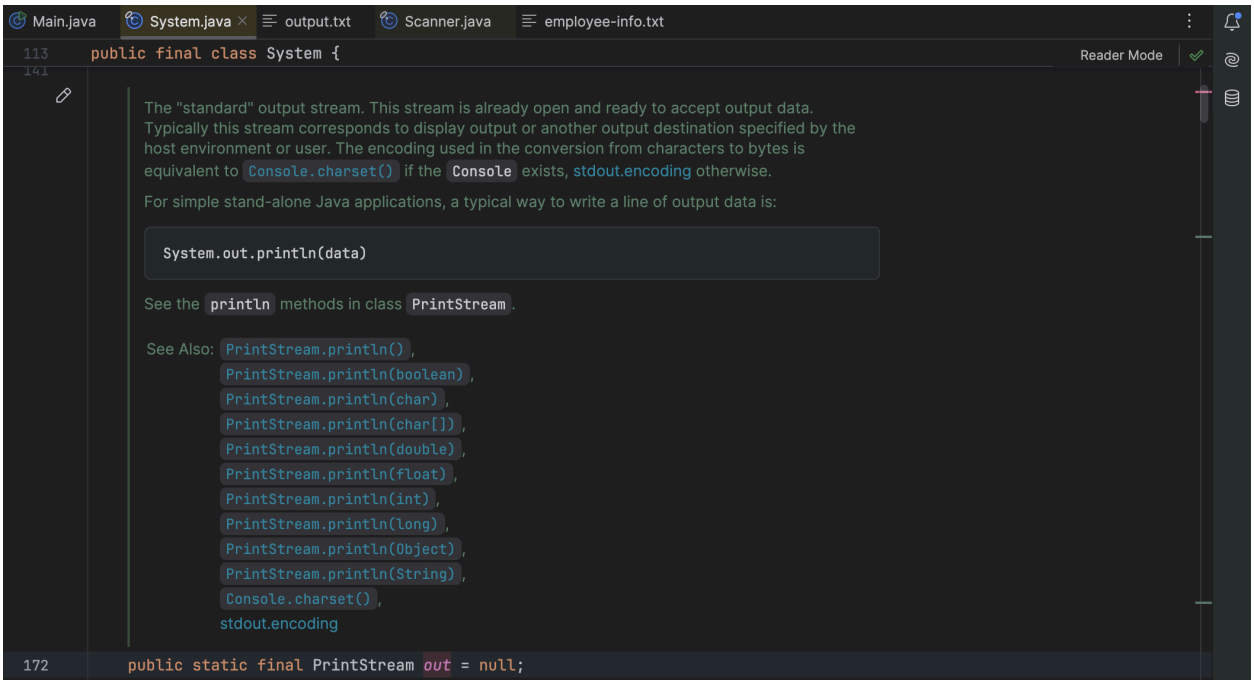
```
PrintStream output;

try {
    output = new PrintStream(file);
} catch (FileNotFoundException e) {
    System.out.println("cannot find \"" + file.getAbsolutePath() + "\"");
    return;
}
```

حالا، می‌تونید دقیقا با همون توابعی که با اون‌ها به کاربر خروجی می‌دادین، توی fileتون بنویسید:

```
output.println("Hello World!");
output.print("This is the last line");
```

شباهت آبجکت PrintStreamای که درست کردین و System.out یک علتی داره، بیاید نگاهی به کد کلاس System بندازیم تا بفهمیم چرا.

توی IntelliJ عبارت System.out رو بنویسید، روی out کلیک راست کنید، به Go To برین و Declaration or Usages رو بزنید تا به جایی که فیلد out تعریف شده برین:



می‌بینید که خود فیلد out هم از جنس PrintStreamئه! هر کاری که با System.out می‌کردین، با متغیر output کد خودتون هم می‌تونید بکنید تا توی یک فایل چاپ کنید.

ورودی و خروجی در کلاس System

الآن که توی سورس کد کلاس Systemاید، وقت خوبیه تا نگاهی به فیلدهای in و out که تا الآن خیلی ازشون استفاده کردین بندازیم. اگر توی کد این کلاس بالا و پایین بشین و کامنت‌های این دو فیلد رو بخونین، می‌تونید تعریف این فیلدها رو ببینید:

```
public final class System {
    // Code Here...

    public static final InputStream in = null;

    public static final PrintStream out = null;

    public static final PrintStream err = null;

    // Code Here...
}
```

می‌بینید که in، صرفا فیلدی از جنس InputStreamئه[1]، و همون‌طور که قبلا دیدین کلاس out هم نوعی PrintStreamئه. علاوه بر این دو کلاس، می‌بینید که کلاس System فیلدی مشابه فیلد out، به اسم err داره. از این فیلد می‌تونید برای خروجی دادن پیغام خطاهاتون به کاربر استفاده کنید.
مثلا:

```
try {
    // Something can throw an exception here
} catch (Exception e) {
    System.err.println(e.getMessage());
}
```

خوبه که همیشه خطاهاتون رو به جای System.out به System.err بنویسید.

[1] کلاس InputStream پدر خیلی از کلاس‌هاییه که برای خوندن ورودی از جاهای مختلف به کار می‌ره. برای این که بیشتر راجع به اون یاد بگیرین، به بخش «منابع بیشتر» مراجعه کنید.

چیزی که یاد گرفتیم

توی این داک، ما مقدمات کار با فایل توی جاوا رو یاد گرفتیم. فهمیدیم که:

- کلاس File چیه، چه کاربردی داره و چطور می‌شه ازش استفاده کرد.
- چطور می‌شه از فایل‌ها خوند و توی اون‌ها نوشت.

منابع بیشتر

چیزی که ما توی این داک یاد گرفتیم، فقط مقدماتی کلی برای کار با فایل‌ها و به طور کلی I/O (مخفف input و output) توی جاوا بود. جاوا امکانات خیلی گسترده‌تری برای مدیریت انواع ورودی و خروجی ارائه می‌ده. اگر دوست دارین بیشتر یاد بگیرین، می‌تونین به فصل دهم کتاب Learning Java، یعنی "File Input and Output" نگاهی بندازین.