کارگاه برنامه نویسی پیشرفته

دستور کار شماره هشت

اهداف

آشنایی با اکسپشن‌ها
آشنایی با پرتاب کردن اکسپشن‌ها
مدیریت خطاها و اکسپشن‌ها با try-catch
انواع اکسپشن‌ها
ساختن اکسپشن اختصاصی خودمان
آشنایی با IO استریم‌ها
معرفی FileStream و ObjectStream
آشنایی با try with resources

فهرست مطالب

اکسپشن‌ها[1]

منظور از اکسپشن‌ها، رخدادهای غیرمنتظره و ناخواسته‌ای است که جریان و روند اجرای طبیعی برنامه را مختل می‌کنند و باعث توقف اجرای برنامه می‌شوند. برای مثال، فرض کنید یک برنامه برای ثبت‌نام تعدادی دانش‌آموز در سامانه‌ای نوشته‌اید، که این برنامه شامل متدی برای دریافت کد ملی دانش‌آموزان است:

```
public class Main {
    public static void main(String[] args) {
        Student student = new Student(null);
    }
}

class Student {
    private String id;

    public Student(String id) {
        if (id == null) {
            throw new IllegalArgumentException("ID is null");
        } else {
            this.id = id;
        }
    }
}
```

در شرایطی که ورودی این متد null باشد برنامه با مشکل مواجه می‌شود.
در چنین شرایطی، متد یک شی از نوع اکسپشن IllegalArgument و با پیغام مورد نظر پرتاب[2] کرده و اجرای برنامه را متوقف می‌کند.

```
Exception in thread "main" java.lang.IllegalArgumentException Create breakpoint : ID is null
    at com.company.Main.register(Main.java:15)
    at com.company.Main.main(Main.java:8)
```

همانطور که مشاهده می‌کنید "Test" چاپ نشد و برنامه با دریافت اکسپشن در متد register، متوقف شد. برای جلوگیری از متوقف شدن برنامه هنگام رخ دادن اکسپشن‌ها نیاز به هندل[3] کردن آن‌ها داریم و برای این کار از try-catch استفاده می‌کنیم.

[1] exception
[2] throw
[3] handle

Try-catch

در بلوک try قسمتی از کد را قرار می‌دهیم که اکسپشن می‌تواند رخ دهد. در بلوک catch نیز عملیات مورد نظرمان برای مدیریت کردن این اکسپشن را قرار می‌دهیم. همچنین نوع اکسپشن‌های مورد نیاز برای هندل کردن باید به عنوان ورودی به catch داده شود.

```
public static void main(String[] args)  {
   try {
       register(null);
   } catch (IllegalArgumentException e) {
       System.out.println("Exception handled...");
   }
   System.out.println("normal flow...");
}
```

```
Exception handled...
normal flow...
```

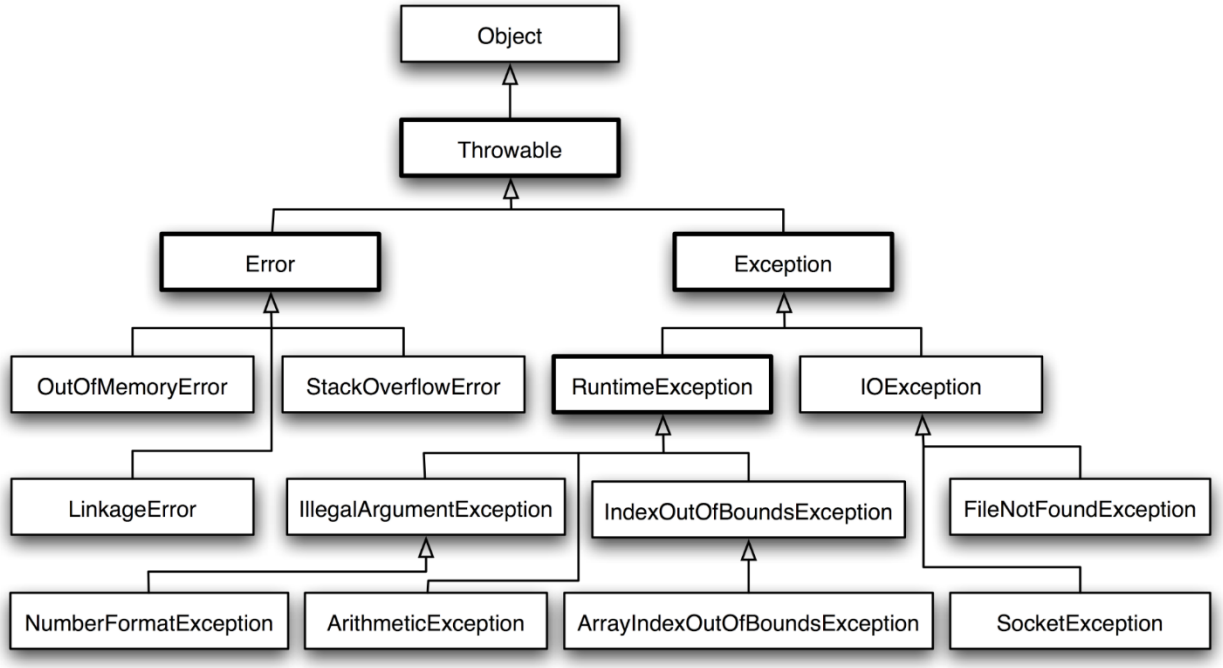
همانطور که مشاهده می‌کنید، این‌بار اکسپشن تولید شده، باعث توقف برنامه نشد.

برای catch می‌توان از چندین اکسپشن هم استفاده کرد[1]:

```
catch (IllegalArgumentException | ArithmeticException e)
```

[1] multi-catch

اکسپشن‌های چک شده[1] و چک نشده



- RuntimeException و اکسپشن‌هایی که از آن ارث‌بری می‌کنند، از نوع چِک نشده محسوب می‌شوند و مربوط به مشکلات در منطق برنامه هستند. در نتیجه اکسپشن‌هایی غیر پیش‌بینی شده به شمار می‌روند.
- اکسپشن‌هایی که از کلاس RuntimeException ارث‌بری نمی‌کنند و در عوض از خود کلاس Exception به صورت مستقیم ارث‌بری می‌کنند، چِک شده به شمار می‌روند و مدیریت نکردن این اکسپشن‌ها، موجب Compile Error می‌شود. این نوع از اکسپشن‌ها قابل پیش‌بینی هستند، مانند: FileNotFoundException

با استفاده از همین ارث‌بری‌ها می‌توانیم اکسپشن‌های مورد نیاز خودمان را هم بسازیم. فرض کنید برنامه‌ای برای ثبت نمرات دانش آموزان ساخته‌اید و می‌خواهید در صورتی که دانش آموزی در لیست یافت نشد یک اکسپشن پرتاب بشود (از جنس چِک شده):

[1] Checked Exceptions

```
class StudentNotFoundException extends Exception {
   public StudentNotFoundException(String message) {
       super(message); // When Exception occured }
  }
}
```

برای مطالعه بیشتر درباره ساختن اکسپشن‌ها می‌توانید به لینک مقابل مراجعه نمایید:

https://stackify.com/java-custom-exceptions

برای مدیریت کردن اکسپشن‌ها دو راه حل وجود دارد:

۱- استفاده از try-catch

۲- استفاده از throws در signature متد مورد نظر

Throws

با استفاده از throws مشخص می‌کنیم که در متد چه اکسپشن‌هایی ممکن است پرتاب شود و در حقیقت وظیفه مدیریت کردن این اکسپشن‌ها (با try-catch) به فراخواننده این متد سپرده می‌شود (که می‌تواند یک متد دیگر و یا خود JVM باشد؛ که در حالت دوم، اگر اکسپشن رخ بدهد برنامه متوقف می‌شود و بازیابی وجود ندارد).

نکته: کلمه throws برای متدهایی که اکسپشن از نوع چِک شده پرتاب می‌کنند حتما باید استفاده شود. استفاده از throws برای اکسپشن‌های چِک نشده الزامی ندارد.

```
public static void main(String[] args) throws IOException {
   register(null);
}
public static void register(String id) throws IOException {
   if(id==null)
   {
       throw new IOException("ID is null");
   }
   else
   {
       //...
   }
}
```

در این حالت چون متد دیگری وجود ندارد که متد main را صدا بزند، IOException پرتاب می‌شود و چون با try-catch هندل نشده است، منجر به توقف برنامه می‌شود.

```
Exception in thread "main" java.io.IOException Create breakpoint : ID is null
    at com.company.Main.register(Main.java:15)
    at com.company.Main.main(Main.java:10)
```

```java
public static void main(String[] args)  {
   try {
       register(null);
   } catch (IOException exception) {
       System.err.println("null id in register method");
   }
}
public static void register(String id) throws IOException {
   if(id==null)
   {
       throw new IOException("ID is null");
   }
   else
   {
       //...
   }
}
```

```
null id in register method
```

(حالت هندل شده)


Java I/O

همان‌طور که می‌دانید، تمام متغیرها و اشیاء که در طول برنامه از آن‌ها استفاده می‌کنیم، در حافظه رم[1] ذخیره می‌شوند و همچنین می‌دانیم که رم یک حافظه پرسرعت اما فرار[2] است. در نتیجه هنگامی که برنامه‌ای را ببندیم یا کامپیوتر را خاموش کنیم، تمام داده‌های ذخیره شده روی آن حذف خواهند شد و امکان دسترسی مجدد به آن‌ها را نخواهیم داشت. اما خیلی وقت‌ها نیاز است که مطالب به صورت بلند مدت ذخیره شوند. در این مواقع باید داده‌های خود را به صورت فایل و بر روی یک حافظه ثانویه[3] مانند HDD ذخیره کنیم. در جاوا این کار توسط java io صورت می‌پذیرد. همچنین تمام کلاس‌های مورد نیاز برای این کار در پکیج java.io موجود هستند.

Stream

در جاوا استفاده از I/O توسط مفهومی به نام استریم[4] صورت می‌گیرد. یک استریم، جریانی از داده‌ها است که بین مبدا و مقصد ساخته شده و باعث انتقال داده‌ها می‌شود.
هنگام اجرا شدن یک برنامه جاوا، سه استریم به صورت خودکار ایجاد می‌شوند که با آن‌ها آشنایی دارید:

- System.in
- System.out
- System.err

به‌طور کلی دو نوع استریم وجود دارد: استریم‌های مبنی بر بایت[5] که ورودی و خروجی را به صورت باینری انجام می‌دهد و استریم‌های مبنی بر کاراکتر[6] که با توالی از کاراکترها کار می‌کند.
در این دستورکار قصد داریم FileReader و FileWriter را به عنوان استریم‌های کاراکتری، و ObjectInputStream و ObjectOutputStream را به عنوان استریم‌های باینری شرح دهیم.

[1] RAM
[2] volatile
[3] secondary storage
[4] stream
[5] byte-based stream
[6] character-based bytes

FileReader و FileWriter

توجه کنید که در جاوا دو نوع استریم ورودی و خروجی داریم و نمی‌توانیم تنها با یک استریم هر دو عمل را انجام دهیم. در صورت نیاز به خواندن و نوشتن به طور همزمان در یک فایل، باید دو استریم مجزای ورودی و خروجی برای آن فایل ایجاد کنیم که هر کدام از این استریم‌ها نشان‌گر فایل متفاوتی دارند. FileWriter نقش خروجی و FileReader نقش ورودی را خواهد داشت.

هنگامی که می‌خواهیم کار با یک فایل را شروع کنیم کافیست یک استریم از نوع مورد نظر بسازیم و در سازنده[1] آن، فایل مورد نظر را با استفاده از آدرس یا یک شئ از نوع File صدا بزنیم. در این حالت استریم مورد نظر ساخته خواهد شد و فایل ما آماده نوشتن یا خواندن خواهد بود.

فرض کنید می‌خواهیم یک فایل متنی با فرمت txt. ایجاد کنیم و داخل آن متنی کوتاه را بنویسیم. با دقت کد زیر را مطالعه کرده و در ادامه به نکات گفته شده توجه کنید:

```
import java.io.FileWriter;
import java.io.IOException;

public class Main {
   public static void main(String[] args) {

       // You must handle the exception of opening a file.
       try {
           FileWriter fileWriter = new FileWriter("test.txt");

           // You can write a string, char or array of chars with FileWriter
           fileWriter.write("This is a test string. \nThis is a new line.");

           // You must close your streams at the end
           fileWriter.close();

       } catch (IOException e) {
           e.printStackTrace();
       }
   }
}
```

۱. بعد از استفاده از یک استریم باید حتما آن را با متد close ببندید. در غیر این صورت تضمینی برای ذخیره شدن داده‌ها به طور صحیح وجود نخواهد داشت.

۲. باز کردن یک فایل، اکسپشن از نوع چِک شده ایجاد می‌کند. به همین دلیل باید مدیریت اکسپشن مناسب انجام شود.

[1] constructor

۳. اگر فایلی را به قصد نوشتن باز کنیم و این فایل از قبل وجود داشته باشد، تمام داده‌های آن از بین خواهد رفت[1] (مگر آنکه فایل را در حالت «append» باز کنیم). به عنوان مثال فرض کنید در کد فوق، از قبل فایلی به نام test.txt در دایرکتوری موجود باشد که شامل یک متن است. پس از اجرای برنامه بالا، تمام متن قبلی آن فایل پاک شده و تنها متن مشخص شده در برنامه باقی خواهد ماند. در ادامه، راه‌حلی برای این مشکل ارائه شده است.

٤. استریم FileWriter دارای یک متد با نام write است که وظیفه آن نوشتن داده‌ی کاراکتری است و می‌توانیم به این متد داده‌هایی از نوع رشته[2]، کاراکتر[3] و آرایه‌ای از کاراکتر[4] پاس دهیم.

حال فرض کنید یک فایل با نام test.txt در دایرکتوری موجود است و متنی هم در آن نوشته شده است. قصد داریم با استفاده از FileReader محتوای این فایل را بخوانیم. به کد زیر توجه کنید:

```
import java.io.IOException;
import java.io.FileReader;

public class Main {
    public static void main(String[] args) {

        try {
            FileReader fileReader = new FileReader("test.txt");

            int i;
            while((i = fileReader.read()) != -1) {
                System.out.print((char)i);
            }

            fileReader.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

متد read یک کاراکتر از فایل می‌خواند، اما باید توجه کنید که نوع خروجی آن به صورت int است. اگر مقدار int برابر «۱-»، باشد به انتهای فایل رسیده‌ایم و کار تمام می‌شود؛ در غیر این صورت، برای نمایش کاراکتر خوانده شده کافیست آن را به char تبدیل کنیم.

[1] Data Truncate
[2] string
[3] char
[4] char array

Object Serialization

در جاوا این امکان وجود دارد که یک شئ را به صورت کامل در یک فایل بنویسیم یا از یک فایل بخوانیم. برای این کار ابتدا شئ مورد نظر باید به صورت توالی از بایت‌ها نوشته شود که شامل اطلاعات شئ مورد نظر و همچنین اطلاعات کلاس آن می‌شود. به این فرآیند در جاوا object serialization گفته می‌شود. هنگام نوشتن شئ در فایل، آن شئ باید serialized شود و هنگام خواندن آن، باید شئ deserialized شود.

Serializable

یک اینترفیس است که هیچ متدی برای پیاده سازی ندارد و صرفا برای نشانه‌گذاری کلاس‌هایی به کار می‌رود که قرار است در فایل ذخیره شوند. کلاسی که قصد داریم اشیاء آن را در فایل ذخیره کنیم باید این اینترفیس را پیاده‌سازی کند در غیر این صورت امکان ذخیره اشیاء وجود نخواهد داشت. توجه کنید که علاوه بر خود شئ، تمام فیلدهای آن هم که نیاز است در فایل ذخیره شوند (ممکن است نیاز به ذخیره بعضی فیلدها نداشته باشیم)، باید serializable باشند.

نکته: تمام داده‌های نوع اولیه به صورت پیشفرض serializable هستند. بقیه کلاس‌ها باید به صورت جداگانه بررسی شوند که برای این کار می‌توانید به مستندات جاوا مراجعه کنید.

در کد زیر قصد داریم یک کلاس با نام Person بسازیم که شامل اطلاعات یک فرد می‌شود و آن را serializable کنیم.

```
import java.io.Serializable;

public class Person implements Serializable {

    private String name;
    private String id;
    private int age;

    public Person(String name, String id, int age) {
        this.name = name;
        this.id = id;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Name: " + name + "\nID: " + id + "\nAge: " + age;
```

```
    }
}
```

ObjectOutputStream

این کلاس برای نوشتن اشیاء serializable داخل فایل استفاده می‌شود. سازنده آن، یک استریم را، که زیرکلاس OutputStream است، دریافت می‌کند. در این مثال از FileOutputStream استفاده می‌کنیم.
کلاس FileOutputStream فایلی که قرار است اشیاء را داخل آن بنویسیم، مشخص می‌کند.

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class Output {
    public static void main(String[] args) {

        Person p = new Person("Ali", "123456789", 24);

        try {
            FileOutputStream fOut = new FileOutputStream("test.bin");
            ObjectOutputStream out = new ObjectOutputStream(fOut);

            out.writeObject(p);

            fOut.close();
            out.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

در این مثال پسوند bin یک پسوند اختیاری است که معمولا برای فایل‌های باینری مورد استفاده قرار می‌گیرد.
متد writeObject کار نوشتن شئ p را در فایل مورد نظر انجام می‌دهد. پس از اجرای برنامه بالا یک فایل با نام test.bin ایجاد خواهد شد که شامل اطلاعات شئ p است.

ObjectInputStream

این کلاس برای deserialize کردن و خواندن اشیاء استفاده می‌شود. کانستراکتور این کلاس، یک استریم را که زیرکلاس InputStream است، دریافت می‌کند. ما در این مثال از FileInputStream استفاده می‌کنیم.
در کد زیر قصد داریم اطلاعات p را که در کد قبل داخل test.bin نوشتیم deserialize کرده و اطلاعات آن را نمایش دهیم.

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class Input {
    public static void main(String[] args) {

        try {
            FileInputStream fIn = new FileInputStream("test.bin");
            ObjectInputStream in = new ObjectInputStream(fIn);

            Person p = (Person)in.readObject();

            System.out.println(p);

            fIn.close();
            in.close();

        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

توجه کنید که خروجی متد readObject یک شئ از نوع Object است. به همین خاطر باید قبل از استفاده، آن را به نوع مورد نظر (در اینجا Person) تبدیل کنیم.

توجه: در یک برنامه ObjectOutputStream حتما باید قبل از ObjectInputStream ساخته شود در غیر این صورت با مشکل مواجه خواهید شد. علت این موضوع در مستندات جاوا شرح داده شده است.

Transient

گاهی اوقات پیش می‌آید که قصد نداریم بخشی از اطلاعات یک شئ را ذخیره کنیم یا به خاطر serializable نبودن بخشی از شئ امکان این کار وجود ندارد. در این موقعیت با استفاده از کلمه کلیدی transient پیش از تعریف فیلد مورد نظر، می‌توانیم باعث serialize نشدن آن شویم.
توجه کنید هنگامی که یک شئ که بخشی از آن transient باشد را از روی فایل می‌خوانیم، مقدار فیلدهای transient شده برابر مقدار پیش فرض خواهد بود.

به نکات مهم زیر در مورد کار با فایل توجه کنید:

۱. شما تنها می‌توانید یک استریم ورودی و یک استریم خروجی به یک فایل داشته باشید که زمانی نیاز می‌شود که بخواهید به طور همزمان در فایل بنویسید و یا از آن بخوانید، در غیراین صورت به طور همزمان به هر دو اینها نیاز نخواهید داشت. ساختن استریم‌های بیش‌تر باعث ایجاد مشکلاتی در کار با فایل‌ها خواهد شد.
۲. امکان استفاده از اشیاء serialize شده توسط برنامه‌های مختلف در پلتفرم‌های متفاوت وجود دارد؛ اما باید این را در نظر بگیرید که کلاسی که می‌خواهید از اشیاء آن استفاده کنید باید در تمام پلتفرم‌ها یکسان باشد. به عنوان مثال اگر کلاس Person که در مثال‌های بالا از آن استفاده کردیم در چند کامپیوتر مختلف وجود داشته باشد، امکان استفاده از فایل ذخیره شده در تمام کامپیوترها وجود دارد.
۳. برای اینکه serialize و deserialize به درستی کار کند، هر کلاس serializable باید یک شماره نسخه مرتبط با آن داشته باشد: *private static final long serialVersionUID*
هدف از این مقدار این است که اطمینان حاصل شود تا کلاس‌های فرستنده (کلاسی که serialize می‌کند) و هم گیرنده (کلاسی که deserialize می‌کند) شیء serialize شده، با یکدیگر سازگار هستند.

Try with Resources

به قطعه کد زیر توجه کنید:

```
try {
   FileOutputStream out = new FileOutputStream(filename);
   String text = "hello world!";
   out.write(text.getBytes());
   out.close();
}
catch (FileNotFoundException e) {
   System.out.println("Could not find the file!");
}
catch (IOException e) {
   System.out.println("I/O error occurred!");
}
```

همانطور که می‌دانید، اگر حین ساختن FileOutputStream یا نوشتن در آن اکسپشن پرتاب شود، وارد یکی از بلاک‌های catch می‌شویم. در این حالت، مابقی بلاک try اجرا نمی‌شود و استریم ما بسته نخواهد شد!

برای رفع این مشکل، می‌توان از بلاک finally استفاده کرد:

```
FileOutputStream out = null;
try {
   out = new FileOutputStream(filename);
   String text = "hello world!";
   out.write(text.getBytes());
}
catch (FileNotFoundException e) {
   System.out.println("Could not find the file!");
}
catch (IOException e) {
   System.out.println("I/O error occurred!");
}
finally {
   if (out != null) {
       try {
           out.close();
       } catch (IOException e) {
           System.out.println("I/O error occurred while closing
the stream!");
       }
   }
}
```

اما بلاک finally چیست؟ این بلاک تحت هر شرایطی از جمله:

۱. اجرا شدن کل بلاک try و رخ ندادن اکسپشن

۲. catch شدن یک اکسپشن

۳. وجود دستور return در هر یک از بلاک‌های try-catch

اجرا خواهد شد.

همانطور که مشاهده می‌کنید، در این حالت try-catch دیگری به کد اضافه شده و آن را طولانی‌تر و دیباگ کردن آن را سخت‌تر کرده است.

نکته قابل توجه دیگر این است که هنگام بستن یک استریم که از قبل بسته شده، IllegalStateException پرتاب می‌شود. از این رو همیشه باید حواسمان باشد که فایلی که قبلاً مثلاً در بلاک try بسته شده است را در finally مجدداً ببندیم.

try with resources تمام مشکلات فوق، از جمله بستن استریم‌ها را حل می‌کند و به ما این اطمینان را می‌دهد که JVM حتماً بعد از اتمام بلاک try فایل مربوطه را می‌بندد. در این مکانیزم، یک یا چند منبع[1] را مانند شکل زیر داخل دستور try اعلام می‌کنیم:

```
try (FileOutputStream out = new FileOutputStream(filename)) {
   String text = "hello world!";
   out.write(text.getBytes());
}
catch (FileNotFoundException e) {
   System.out.println("Could not find the file!");
}
catch (IOException e) {
   System.out.println("I/O error occurred!");
}
```

حال اگر اکسپشن پرتاب و یا بلاک try به طور کامل اجرا شود، منابع به صورت خودکار بسته می‌شوند (دلیل: AutoClosable).

[1] resource

نکته

در مثال‌های زیر، دقت کنید که Exceptionهای خود را به صورت جزئی catch یا پرتاب کرده و سعی کنید از کلاس‌های کلی مثل Exception و یا IOException کمتر استفاده کنید.

الف) در تعریف متد[1]ها و هنگام پرتاب کردن اکسپشن:

فرض کنید که هم‌تیمی شما در یک پروژه تعدادی متد تعریف کرده و شما می‌خواهید از این متدها استفاده کنید. بدیهی است که تعریف متد اول، برای شما واضح نخواهد بود. تعریف صحیح، تعریف متد دوم است که در آن نام اکسپشن مورد نظر، ذکر شده است.

```
public void doNotDoThis() throws Exception {

}

public void doThis() throws IllegalArgumentException {

}
```

ب) برای catch کردن اکسپشن‌ها، دو روش زیر را در نظر بگیرید:

روش اول)

```
try (FileOutputStream out = new FileOutputStream(filename)) {
   String text = "some text";
   out.write(text.getBytes());
}
catch (Exception e) {
   e.printStackTrace();
}
```

روش دوم)

```
try (FileOutputStream out = new FileOutputStream(filename)) {
   String text = "some text";
   out.write(text.getBytes());
}
catch (FileNotFoundException e) {
   System.out.println("Could not find the file!");
   // do a specific task (eg: try to open another file, ...)
}
catch (IOException e) {
   System.out.println("I/O error occurred!");
```

[1] Method Signature

```
    // do a specific task (eg: display an error message to the user, ...)
}
```

واضح است که در روش دوم، نه تنها خوانایی کد بالاتر می‌رود، بلکه شما می‌توانید اکسپشن‌های احتمالی برنامه خود را حالت‌بندی کرده و برای هر کدام، راه‌حل‌های متفاوتی[1] ارائه دهید. در این روش به ترتیب catch کردن اکسپشن‌ها دقت کرده و با توجه به اصول وراثت و سلسله مراتب کلاس‌های اکسپشن، ابتدا اکسپشن‌های فرزند را catch کنید.

دفترچه یادداشت

در این جلسه قصد داریم یک دفترچه یادداشت شخصی در محیط کنسول، پیاده‌سازی کنیم.

فراموش نکنید که کدهای خود را بر روی یک ریپازیتوری با نام «AP-Workshop8» قرار دهید.

این دفترچه یادداشت، باید قابلیت‌های زیر را داشته باشد:

- اضافه کردن یک یادداشت جدید
- حذف کردن یک یادداشت دلخواه
- مشاهده لیستی از تمام یادداشت‌ها
- گرفتن خروجی از یک یادداشت دلخواه

```
1- Add
2- Remove
3- Notes
4- Export
```

در ادامه برای هر بخش، توضیحاتی خواهیم داد.

[1] recovery Actions

Add

- کاربر بتواند برای هر یادداشت جدید یک اسم دلخواه انتخاب کند (اگر اسم انتخاب شده تکراری بود، عکس العمل مناسبی رخ دهد).
- کاربر بتواند یک یادداشت جدید، که می‌تواند شامل یک یا چندین خط باشد، بنویسد.
- یادداشت جدید ایجاد شده در جایی ذخیره شود؛ به طوری که حتی در اجراهای بعدی، کاربر بتواند به به آن دسترسی داشته باشد.

انتظار می‌رود کاربر فقط از طریق اجرای برنامه بتواند به یادداشت‌ها دسترسی داشته باشد. پس توجه کنید که یادداشت‌ها نباید در فایل txt ذخیره شوند!

- تاریخ ایجاد یادداشت در جایی به همراه یادداشت ذخیره شود (ذخیره روز، ماه و سال ایجاد یادداشت کافیست).

```
1- Add
2- Remove
3- Notes
4- Export
1
please choose a title for the note:
Alan Turing
ok. feel free to write!
enter '#' to finish!
Alan Mathison Turing OBE FRS was an English
mathematician, computer scientist, logician,
cryptanalyst, philosopher, and theoretical biologist.
#
the new note has been added successfully!
```

Remove

- لیستی از یادداشت‌های ذخیره شده نمایش داده شود و کاربر با انتخاب یکی از یادداشت‌ها، بتواند آن را حذف کند.

```
1- Add
2- Remove
3- Notes
4- Export
2
choose one of the notes to remove or enter '0' to back to main menu:
1- Alan Turing      2021-08-30
2- AP_Workshop      2021-08-30
3- remove_me        2021-08-30
3
the note has been removed successfully!
```

Notes

- لیستی از تمام یادداشت‌های ایجاد شده نمایش داده شود. سپس کاربر با انتخاب یک یادداشت دلخواه، بتواند محتوای داخل آن را مشاهده کند.

```
1- Add
2- Remove
3- Notes
4- Export
3
choose a note to show:
1- Alan Turing      2021-08-30
2- AP_Workshop      2021-08-30
3- remove_me        2021-08-30
1
----        Alan Turing     ----

Alan Mathison Turing OBE FRS was an English
mathematician, computer scientist, logician,
cryptanalyst, philosopher, and theoretical biologist.
```

Export

- لیستی از تمام یادداشت‌های ایجاد شده نمایش داده شود و کاربر بتواند یکی از یادداشت‌ها را برای خروجی گرفتن انتخاب کند. سپس، از یادداشت انتخاب شده، یک خروجی گرفته شود و در یک فولدر مناسب (برای مثال export) ذخیره شود. خروجی باید یک فایل txt باشد. اسم فایل txt نیز همان عنوان یادداشت است.

```
1- Add
2- Remove
3- Notes
4- Export
4
choose a note to export:
1- Alan Turing      2021-08-30
2- AP_Workshop      2021-08-30
3- remove_me        2021-08-30
1
the note has been exported successfully!
you can find It on "export" directory.
```

نکات کلی:

- با توجه به اینکه در جلسه قبل، اصول طراحی را آموختید، طراحی این برنامه نیز به عهده شما خواهد بود.
- سعی کنید از تمام آموخته‌های خود در این جلسه، برای پیاده‌سازی این تمرین استفاده کنید.

هدف این تمرین علاوه بر ایجاد تسلط نسبی شما بر روی کار با فایل، Exception handling نیز می‌باشد؛ پس انتظار می‌رود ورودی‌های اشتباه کاربر را نیز هندل کنید.

- توجه کنید که در هر یک از چهار قسمت بالا، کاربر باید بتواند به صفحه اصلی بازگردد.

برای مثال، اگر کاربر وارد قسمت Add شد، درصورت پشیمانی، بتواند به صفحه اصلی بازگردد، بدون آنکه یادداشت جدیدی اضافه کند.

- لازم به ذکر است که برنامه تمام نخواهد شد، مگر آنکه کاربر اقدام به بستن برنامه کند (به این منظور، می‌توانید یک گزینه «exit» نیز به برنامه اضافه کنید).
- درضمن، عکس‌هایی که مشاهده کردید صرفا یک نمونه پیاده‌سازی برای فهم بهتر سوال بود. شما مختارید که پیاده‌سازی را به هر نحو که دوست دارید انجام دهید. فقط باید شروط کلی را رعایت کنید.