آموزش زبان برنامه نویسی جاوا

(استثناها) Exceptions

جلسه بیست و نهم

نویسنده : رحمان زارعی

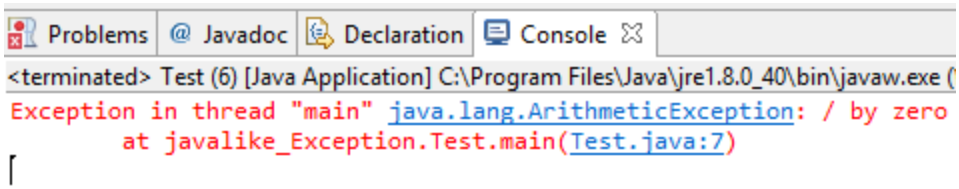
جاوا را ساده،آسان و شیرین بنوشید!!!!

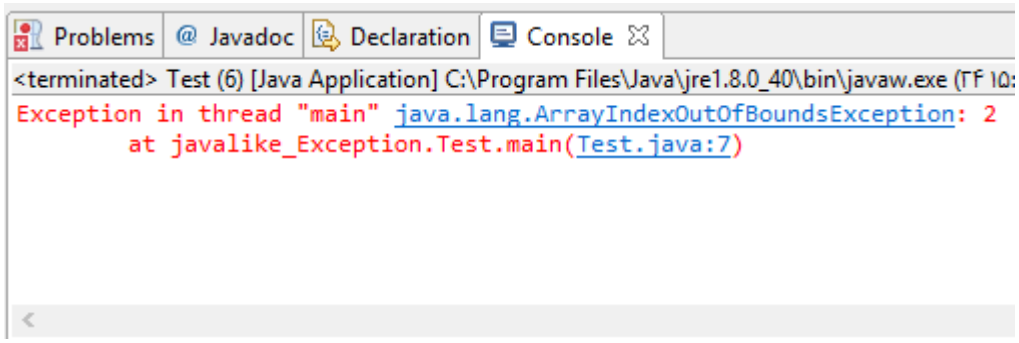
به Exception در فارسی استثنا می گویند.

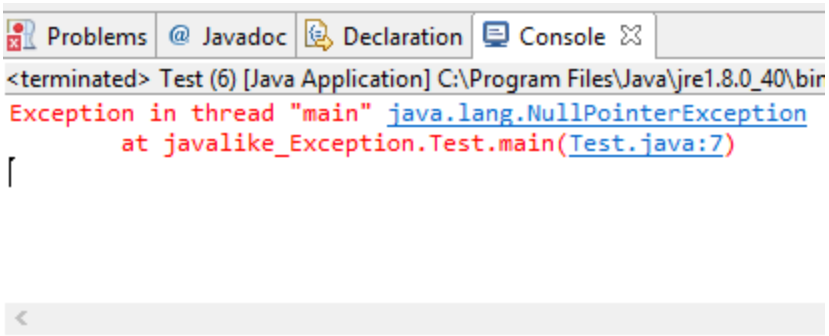
Exception (استثنا) چیست؟ یک exception در برنامه نویسی مشکلی است که در طول اجرای یک برنامه رخ می دهد.هنگامی که یک استثنا یا اکسپشن (exception) رخ می دهد ، جریان طبیعی برنامه مختل و برنامه بصورت غیر طبیعی خاتمه می یابد.

در کل و به زبان ساده به خطاهای قابل کنترل و تصحیحی که هنگام اجرای برنامه رخ می دهد و باعث توقف جریان طبیعی برنامه می شود Exception (استثنا) می گوییم.

شما شاید با مفهوم استثنا تازه دارید آشنا می شوید اما احتمالا حداقل یکبار هنگام اجرای برنامه با خطاهای زیر برخورد کرده اید!!!!







همه این تصاویر بالا گویای رخ دادن استثنا هنگام اجرای برنامه جاوا در Eclipse هستش که به این خطاها Exception (استثنا) می گوییم.

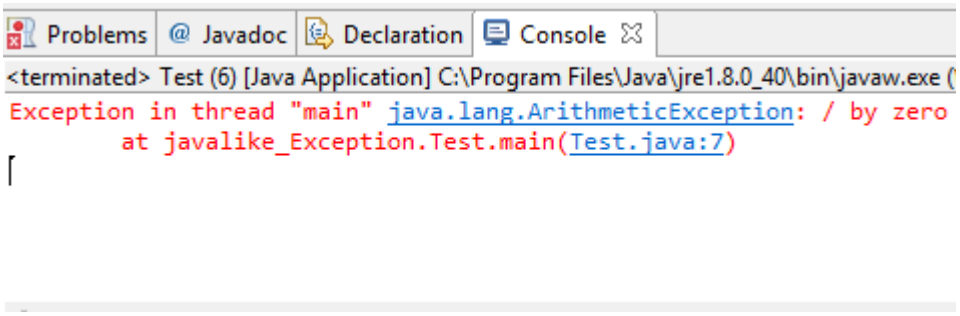
خب جاوا برای کنترل این Exceptions (استثناها) ابزارهایی را پیش بینی کرده است!!

برخی از Exceptions (استثناها) خیلی معروف هستند از بس که اتفاق می افتند و برخی هم کمتر معروف اند:

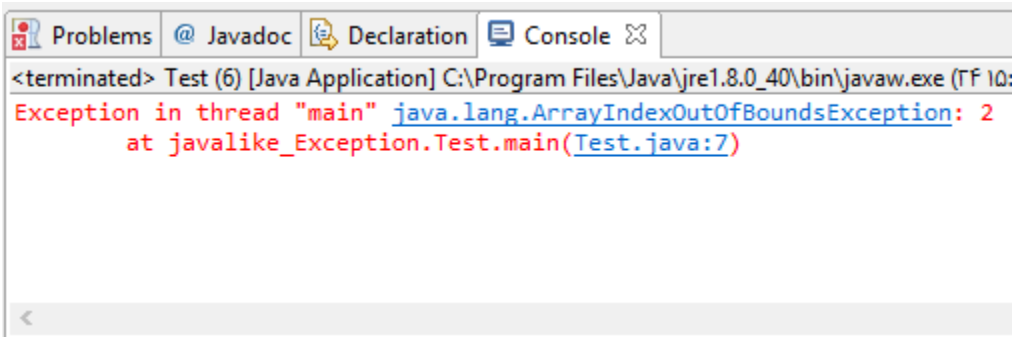
مثلا:

- Exceptions (استثناهای) زیر مربوط به خطاهایی است که در عملیات ریاضی احتمال دارد دهند:

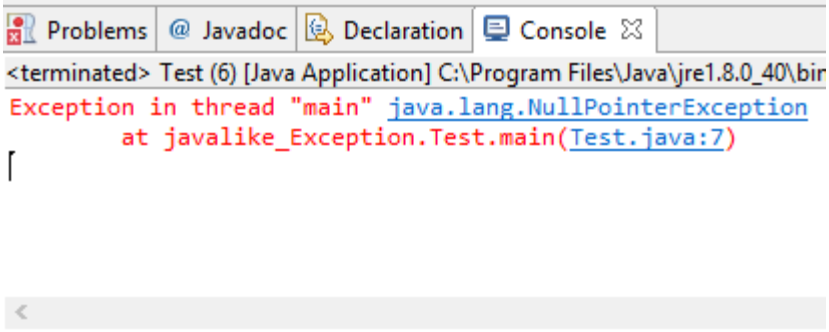
مثلا اگر یک عدد را بر صفر تقسیم کنیم جواب مبهم است،در برنامه نویسی نیز اگر یک عدد را بر صفر تقیسم کنیم Exception (استثنا) زیر اتفاق می افتد.



- Exception (استثنا) زیر مربوط به اندیس های خانه آرایه می باشد،وقتی که خانه آرایه ای با اندیس یا شماره ای بیشتر تعداد خانه های آرایه یا طول آرایه صدا می زنیم.



- اگر آرایه ای را صدا بزنیم که مقدار عناصر درون آن پوچ یا null باشد Exception (استثنا) زیر رخ می دهد:



ما Exceptions (استثناهای) زیادی در جاوا داریم و قصد نداریم به معرفی انواع استثناها در جاوا بپردازیم و هدف از مثالهای بالا اشنایی و درک بهتر Exceptions (استثناها) در جاواست.

خب جاوا دستورالعمل هایی پیش بینی کرده که می توانیم با کمک آنها در جاهایی از کدهای برنامه نویسی خود که احتمال رخ دادنExceptions (استثناها) را می دهیم این استثناها را کنترل و مدیریت کنیم.یعنی با مدیریت استثناها حداقل باعث شویم که اگر خطایی رخ داد ،روند اجرای طبیعی برنامه را مختل و متوقف نکند.

علتّ های رخْ دادنْ Exceptions (استثْناها) :

یک استثنا به دلایل مختلفی ممکن است رخ دهد.در زیر برخی از حالاتی که در آن Exceptions (استثناها) رخ میدهد آورده شده است:

- وقتی که کاربر داده های نامعتبر و بی ربط وارد کند.

مثلا متغیر ما از نوع رشته است اما کاربر یک عدد وارد می کند.

- وقتی ما میخواهیم یک فایل را باز و بخوانیم اما آن فایل در سیستم وجود نداشته باشد. (در جلسات آینده کار با فایل را خواهیم آموخت)
- Null یا پوچ بودن مقدار متغیرها
- عملیات ریاضی که جواب منطقی ندارد مثل تقسیم عدد بر صفر
- انتخاب اندیسی از آرایه که بیشتر از طول آرایه است
- و.....

برخی از استثناها توسط کاربر یعنی کسی که از برنامه استفاده می کند رخ می دهد، برخی توسط خود برنامه نویسی که برنامه رو نوشته رخ می دهد و برخی دیگر توسط منابع فیزیکی رخ میدهد.

در کل Exceptions (استثناها) را به سه دسته می توان تقسیم کرد،که با درک و فهمیدن آنها می توانید Exceptions (استثناها) برنامه خود را مدیریت و کنترل کنید.

Checked exceptions ():

استثناها و خطاهایی که زمان کامپایل برنامه اتفاق می افتد و استثنای زمان کامپایل (compile time exceptions) نامیده می شود. خب استثنای زمان کامپایل در جاوا دیگه خیلی مشخص هست خود برنامه Eclipse هنگام کدزنی با خطوط قرمز این خطاها را برای ما مشخص کرده و هشدار می دهد و برنامه نویس به سادگی نمی تواند از آن بگذرد و یک برنامه نویس باید این خطاها را کنترل کند.

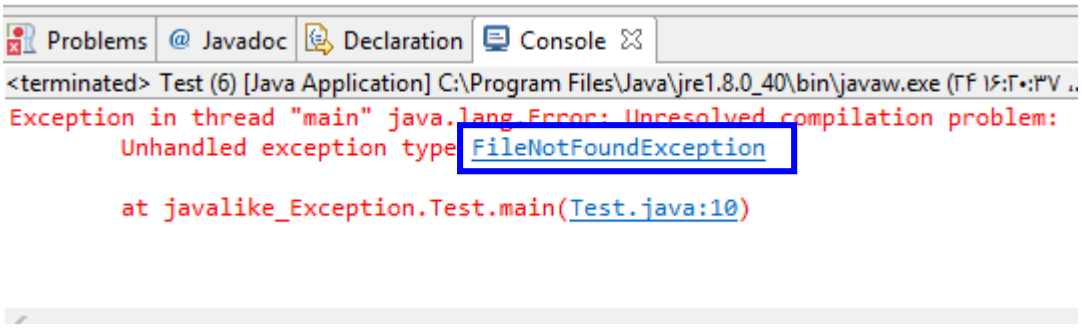
مثالی که برای استثنای زمان کامپایل (compile time exceptions) می توان زد کلاس FileReader هستش که از طریق ان می توانید داده های درون یک فایل را بخوانید، خب امکان داره فایل مربوطه در کامپیوتر شما وجود نداشته باشد و احتمال رخ دادن Exception (استثنا) FileNotFoundException وجود دارد، دیگه برای استثنای زمان کامپایل (compile time exceptions) رخ می دهد یعنی بدون اجرای برنامه ، به شما احتمال رخ دادن خطا هشدار داده خواهد شده و شما به عنوان یک برنامه نویس باید بتوانید این خطا را مدیریت کنید.

```
package javalike_Exception;

import java.io.File;
import java.io.FileReader;

public class Test {

public static void main(String[] args) {
            File file = new File("E://file.txt");
            FileReader fr = new FileReader(file);
      }

}
```

- خط زرد رنگ استثنای زمان کامپایل (compile time exceptions) رخ می دهد.

```
1  package javalike_Exception;
2
3  import java.io.File;
4  import java.io.FileReader;
5
6  public class Test {
7
8      public static void main(String[] args) {
9          File file = new File("E://file.txt");
10         FileReader fr = new FileReader(file);
11     }
12
13 }
14
```

- اگر شما کد بالا را اجرا کنید با خطای زیر برخورد میکنید:



کامپایلر برای محکم کاری میگه باید این استثنا را مدیریت و کنترل کنید که اگر هم فایلی در کامپیوتر وجود نداشت روند اجرای طبیعی برنامه متوقف نشود.

مثال کلاس FileReader را کلا فراموش کنید چون ما هنوز کار با فایل رو شروع نکردیم و تنها گزینه مناسبی برای مثال زدن در مورد استثنای زمان کامپایل (compile time exceptions) بود که زدیم پس اصلا نگران مفاهیم این موضوع کار با فایل نشید چون در جلسه ای جداگانه در آینده قشنگ بررسیش می کنیم ☺

۲) Unchecked exceptions:

Exceptions (استثناهای) که در **زمان اجرای** برنامه اتفاق می افتند، این استثناها ، استثناهای زمان اجرا (Runtime Exceptions) نامیده می شوند. از نمونه باگ های (اشکالات) برنامه نویسی که شامل این نوع استثناها می باشد می توان به خطاهای منطقی که در زمان اجرای برنامه رخ می دهد اشاره کرد.

نکته : به اشکالات و گیرهایی که در برنامه نویسی اتفاق می افتد باگ (bug) می گویند.

برای مثال ، اگر شما یک آرایه را با اندازه ۵ در برنامه خود تعریف کرده باشید و قصد دارید عنصر خانه ۶ آرایه را صدا بزنید برنامه دچار استثنا ArrayIndexOutOfBoundsExceptionexception می شود.

مثال :

```
package javalike_Exception;

public class Test {

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

```
        int num[] = { 1, 2, 3, 4 };
        System.out.println(num[5]);
    }

}
```

خروجی:اگر برنامه را اجرا کنید با استثنای زیر برخورد می کنید.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at javalike_Exception.Test.main(Test.java:10)
```

```
int num[] = { 1, 2, 3, 4 };
```

- اندازه و تعداد خانه های آرایه ۴ می باشد.

```
System.out.println(num[5]);
```

- ما عنصر خانه پنجم آرایه را صدا زده ایم ، از آنجایی که اندازه آرایه ما ۴ می باشد با استثنایی که در خروجی نمایش داده شده برخورد می کنیم.

۳) Errors (خطاها):

همه اشکالات و گیرهایی که هنگام اجرای برنامه اتفاق می افتد استثنا (Exception) نیستند!!!! به طور کلی هر خطایی استثنا نیست!!!! در کل مشکلات و گیرهایی که خارج از کنترل کاربر یا برنامه نویس باشد استثنا (Exception) نیستند و به آنها خطا (Errors) می گوییم. مثل خطای stack overflow ، خطای حافظه (memory error) ، خطای سخت افزار (hardware error) ، خطای ماشین مجازی جاوا (JVM error) و.....

پس فرق خطا با استثنا این است که استثناها در برنامه نویسی قابل کنترل و مدیریت هستند اما خطاها از کنترل برنامه نویس خارج هستند.

Exception خود یک کلاس می باشد که در کتابخانه java.lang.Exception قرار دارد. کلاس Exception فرزند کلاس Throwable است ، دیگه جزییات رو براتون باز نمیکنم در همین حد بدانید کافی است.

برخی از متدهای کلاس Exception :

`public String getMessage()`

این متد جزییات استثنایی که رخ می دهد را برمی گرداند.

`public Throwable getCause()`

این متد علت استثنایی که رخ داده است را برمیگرداند.

کنترل و مدیریت استثناها:

گفتیم فرق استثنا با خطا در این هست که برنامه نویس میتونه استثناها را کنترل و مدیریت کند.خب در اینجا هم قصد داریم یادبگیریم که چطور استثناهایی که رخ می دهند را مدیریت و کنترل کنیم.

try - catch :

شما می توانید با استفاده از ترکیب کلمه های کلیدی `try - catch` یک استثنا را مدیریت و کنترل کنید .

چگونه؟!

اول نحوه نوشتن `try - catch` را بررسی می کنیم:

```
try {

}catch(Exception e) {

}
```

```
try {

}
```

بین دو بلوک آکولاد باز و بسته جلوی کلمه کلیدی `try` بخشی از کد که احتمال رخ دادن استثنا در آن هست را قرار می دهیم.

```
catch(Exception e)
```

درون پرانتز باز و بسته جلوی کلمه کلیدی catch یک شی از نوع کلاس استثنایی که احتمال می دهیم رخ دهد قرار می دهیم.

نکته مهم : هر استثنایی که رخ می دهید یک کلاس می باشد که فرزندان کلاس Exception می باشند!! یعنی مثلا استثنای ArrayIndexOutOfBoundsException خود یک کلاس می باشد که فرزند کلاس Exception می باشد.پس هر استثنا (Exception) که رخ می دهد یک کلاس هستش که فرزند کلاس Exception می باشد.

خب اگر نوع استثنایی که احتمال رخ دادن آن وجود دارد را توانستیم تشخیص بدهیم درون پرانتز جلوی کلمه کلیدی catch از کلاس استثنایی که ممکنه رخ دهد شی می سازیم اگر نتوانستیم تشخیص بدهیم بجای آن می توانیم از کلاس Exception درون پرانتز جلوی کلمه کلیدی catch شی بسازیم ، چون کلاس Exception پدر همه استثناها می باشد مشکلی پیش نمی آید.

پس درون پرانتز جلوی کلمه کلیدی catch بهتر است از کلاس استثنایی که احتمال رخ دادن آن هست شی بسازیم اما اگر نتونستیم تشخیص بدهیم که از کدام کلاس استثنا استفاده کنیم می توانیم از کلاس Exception که پدر همه استثناها هست شی بسازیم.

```
catch(Exception e) {

}
```

بخشی از کد برنامه مون که احتمال رخ دادن استثنا در آن بود رو درون بلوک جلوی کلمه کلیدی try قرار داده ایم، وقتی در این بخش از کد استثنایی رخ دهد برنامه به سراغ کلمه کلیدی catch می رود و پرانتز جلوی کلمه کلیدی catch را بررسی می کند اگر استثنایی که رخ داده از نوع کلاس یا فرزند کلاسی باشد که در پرانتز جلوی کلمه کلید catch از آن شی ساخته ایم باشد خب استثنا کنترل می شود و برنامه دستورات درون بلوک آکولاد جلوی کلمه کلیدی catch را اجرا می کند.

ما می توانیم درون آکولاد جلوی کلمه کلیدی catch یک پیام چاپ کنیم که گویای رخ دادن استثنا باشد یا این که دستور خاصی رو قرار بدیم که اگر استثنایی رخ دادن مثلا فلان دستور اجرا شود و.....

میدونم شاید کاملا متوجه نشده باشید اصلا نگران نباشید با مثال حل کردن درست میشه ☺

```
try {
کدی که احتمال رخ دادن استثنا در آن هست اینجا قرار می دهیم
}catch(Exception e تعریف شی از نوع  کلاس استثنایی که احتمال رخ دادن آن وجود دارد) {
در صورت رخ دادن استثنا دستورات درون این بلوک اجرا می شود
}
```

پس در کل کدی که احتمال رخ دادن استثنا در آن می باشد را درون بلوک کلمه کلیدی try قرار می دهیم و در صورت رخ دادن استثنا کلمه کلیدی catch استثنا را مهار و کنترل می کند و اجازه نمی دهد روند اجرای طبیعی برنامه متوقف شود و بعد از رخ دادن استثنا دستور درون بلوک جلوی کلمه کلیدی catch اجرا می شود.

چه چیزی بهتر است یادگیری با مثال ☺

مثال:

```
package javalike_Exception;

public class Test {

	public static void main(String[] args) {

		int a[] = new int[2];
		System.out.println("Access element three :" + a[3]);

		System.out.println("Hello Javalike");

	}

}
```

خروجی:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
	at javalike_Exception.Test.main(Test.java:8)
```

- خب در این مثال اندازه آرایه a برابر ۲ می باشد، اما در خط بعد ما مقدار عنصر خانه شماره ۳ آرایه را صدا زده ایم که با استثنای اندیس آرایه مواجه شده ایم چون آرایه a اندازه آن ۲ است و خانه شماره ۳ ندارد.
- نکته ای که این مثال دارد بعد از رخ دادن استثنا روند اجرای طبیعی برنامه متوقف شده و خط چاپ پیام "Hello Javalike" اجرا نشده است.

- دلیل مدیریت استثناها هم همین است ما میخواهیم کاری کنیم که اگر استثنایی در مکانی از کدهایمان رخ داد حداقل ضربه ای به روند اجرای سایر دستورات برنامه ما وارد نشود!
- حالا قصد داریم این استثنا را با try-catch کنترل و مدیریت کنیم که برنامه متوقف نشود.و پیام "Hello Javalike" در خروجی حتی با رخ دادن استثنا اجرا شود.

```
package javalike_Exception;

public class Test {

	public static void main(String[] args) {

		int a[] = new int[2];

		try {

			System.out.println("Access element three :" + a[3]);

		} catch (ArrayIndexOutOfBoundsException e) {

			System.out.println("Exception thrown  :" + e);
		}

		System.out.println("Hello Javalike");
	}

}
```

خروجی:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Hello Javalike
```

```
System.out.println("Access element three :" + a[3]);
```

این خط از کد که احتمال رخ دادن استثنا را می دادیم درون بلوک جلوی کلمه کلیدی try قرار داده ایم.

وقتی کامپایلر برنامه این خط کد را اجرا میکند و باعث بروز استثنا به دلیل نبود اندیس عدد ۳ در آرایه a می شود ،برنامه بصورت خودکار به سراغ جناب catch می رود و ازش خواهش میکنه که جون بابات این استثنا رو کنترل کن که برنامه متوقف نشه جناب catch هم از انجایی که وظیفه و تخصص مهار استثناها می باشد قبول میکنه و دو دستی استثنا رو گیر میندازه! ☺ و بعدش دستور درون بلوک جلوی catch اجرا میشه مثلا الکی catch میخواد خودنمایی کنه که من این استثنا رو به

دام انداختم اینم مدرکش ☺ همان طور که در خروجی مشاهده می کنید پیام Hello Javalike چاپ شده است این یعنی استثنا هیچ مانعی از روند اجرای طبیعی برنامه نشده فقط یک خط کد که قصد طغیان رو داشت رو سرکوب کرد ☺

```
catch (ArrayIndexOutOfBoundsException e)
```

چون استثنا از نوع اندیس آرایه هست از کلاس فوق درون پرانتز جلوی catch نمونه ساخته ایم .

در زیر از یکی از متدهای کلاس Exception استفاده کرده ایم که جزییات استثنایی که رخ میدهد را به ما میگه:

```
package javalike_Exception;

public class Test {

	public static void main(String[] args) {
		int a[] = new int[2];
		try {

			System.out.println("Access element three :" + a[3]);
		} catch (ArrayIndexOutOfBoundsException e) {
			System.out.println(e.getMessage());
		}
		System.out.println("Hello Javalike");
	}

}
```

خروجی:

```
3
Hello Javalike
```

- خب میگه که عدد ۳ یکی از دلایل رخ دادن استثنا بوده است.
- ما می توانیم درون بلوک جلوی کلمه کلیدی catch با استفاده از متدهای خود کلاس Exception یا چاپ پیامی دلایل و جزییات استثنا را در خروجی چاپ کنیم یا این که با رخ دادن استثنا به برنامه بگیم یک دستور خاصی را برای ما اجرا کن.

Multiple Catch Blocks (استفاده از چند بلوک catch):

کلمه کلید try می تواند چندین بلوک catch داشته باشد، نحوه پیاده سازی آن بصورت زیر است:

```
try {
        کدی که احتمال رخ دادن استثنا در آن وجود دارد
        }catch(ExceptionType1 e1 تعین نوع استثنای اول ) {
در صورت رخ دادن استثنا مرتبط با این بلوک catch دستورات درون این بلوک اجرا می شود

        }catch(ExceptionType2 e2 تعین نوع استثنای دوم ) {
در صورت رخ دادن استثنا مرتبط با این بلوک catch دستورات درون این بلوک اجرا می شود
        }catch(ExceptionType3 e3 تعین نوع استثنای سوم ) {
در صورت رخ دادن استثنا مرتبط با این بلوک catch دستورات درون این بلوک اجرا می شود
        }
```

- در این دستورات بالا ما از سه بلوک `catch` استفاده کردیم شما می توانید به هر تعداد که خواستید با توجه به نیاز بلوک `catch` اضاف کنید.
- پس شما می توانید برای یک بلوک `try` چندین بلوک `catch` تعریف کنید.
- ترتیب اجرای دستور بالا به این صورت است که بعد از اجرای کد درون بلوک `try` و رخ دادن استثنا برنامه بلوک `catch` ها را بررسی می کند و در صورتی که استثنای رخ داده از نوع یکی از بلوک `catch` ها باشد دستورات درون آن بلوک `catch` اجرا می شود.
- این نوع دستورات برای این هست که وقتی شما چند خط کد دارید و برای این چند خط کد احتمال رخ دادن استثناهای مختلفی وجود دارد.

مثال :

```
package javalike_Exception;

public class Test {

     public static void main(String[] args) {

          int a[] = { 2, 10 };

          try {
               System.out.println("Access element zero Array a:" + a[0]);
               System.out.println("Access element one Array a/0:" + a[1] / 0);
               System.out.println("Access element two Array a:" + a[2]);

          } catch (ArrayIndexOutOfBoundsException e) {
               System.err.println(e.getMessage());
```

```
                System.err.println(e+"\n");

            } catch (ArithmeticException e) {
                System.err.println(e.getMessage());
                System.err.println(e);
            }
            System.out.println("Hello Javalike");
        }

}
```

خروجی:

```
Access element zero Array a:2
java.lang.ArithmeticException: / by zero
/ by zero
Hello Javalike
```

```
System.out.println("Access element one Array a/0:" + a[1] / 0);
                System.out.println("Access element two Array a:" + a[2]);
```

- در این خط از کد احتمال رخ دادن دو استثنای ریاضیات و استثنای اندیس آرایه وجود دارد به همین خاطر دو بلوک catch تعریف کرده ایم.
- برنامه خط به خط کدهای درون بلوک try را اجرا میکند همین که به یک استثنایی برخورد کرد ، دیگه خط های بعدی اجرا نمی شود و برنامه هدایت می شود به سمت یکی از بلوک catch هایی که استثنای رخ داده از نوع ان می باشد.

```
System.err.println(e);
```

اگر به جای کلمه out کلمه err در دستور چاپ در خروجی کنسول قرار دهیم پیام چاپ شده به رنگ قرمز خواهد بود و معمولا برای چاپ پیام هشدار و اخطار و خطا و.... استفاده می شود.

مثال :

```
package javalike_Exception;

public class Test {

    public static void main(String[] args) {

        System.err.println(" Error!!!!!");
```

```
    }
}
```

خروجی:

```
Error!!!!!
```

همان طور که می بینید این پیام به رنگ قرمز در محیط کنسول جاوا چاپ شده است.

نکته مهم : از نسخه JDK 7 به بعد شما می توانید چندین کلاس استثنا را در یک بلوک catch تعریف کنید.با این کار دیگر شما نیاز به تعریف چندین بلوک catch نخواهید داشت.

مثال :

```
package javalike_Exception;

public class Test {

	public static void main(String[] args) {

		int a[] = { 2, 10 };

		try {
			System.out.println("Access element zero Array a:" + a[0]);
			System.out.println("Access element one Array a/0:" + a[1] / 0);
			System.out.println("Access element two Array a:" + a[2]);

		} catch (ArrayIndexOutOfBoundsException | ArithmeticException e) {
			System.err.println(e.getMessage());
			System.err.println(e + "\n");
		}
		System.out.println("Hello Javalike");
	}

}
```

خروجی:

```
Access element zero Array a:2
/ by zero
java.lang.ArithmeticException: / by zero

Hello Javalike
```

همان طور که مشاهده میکنید شما تنها یک بلوک try-catch دارید و در بلوک catch از دو نوع کلاس استثنا استفاده شده است.برای این کار کافیست کلاس های استثناهایی که احتمال رخ دادن آنها وجود دارد را نوشته و هر کدام را با علامت "|" متمایز کرده و در آخر نامی برای شی آنها انتخاب می کنید:

```
catch (ArrayIndexOutOfBoundsException | ArithmeticException e)
```

```
catch (انتخاب نامی برای شی .....|کلاس استثنا دوم|کلاس استثنا اول)
```

کلمات کلیدی Throws/Throw :

معنی فارسی کلمه کلیدی Throws **پرتاب کردن** می باشد.

اگر در یک **متد** نمی خواهید **استثنای** کدهای درون آن را بررسی و رسیدگی کنید،می توانید برای متد خود از کلمه کلیدی Throws استفاده کنید.کلمه کلیدی Throws را می توانید **کنار امضا یا همان پرانتز جلوی متد** قرار دهید.

شما می توانید با استفاده از کلمه کلیدی Throws ، استثنای درون متد خود را پرتاب کنید! و از شرش راحت شوید، خب این یعنی چه؟! برای درک بهتر چه مثالی از دنیای واقعی بزنم که روشن کنه این مفهوم رو؟! الکی یک مثالی میزنم!!! صرفا جهت روشن شدن مفهوم، فرض کنیم داخل بقالی جعفر رفته ایم و یک دونه پفک خریده ایم و بعد از خوردن بفک!!! پوست پفک را داخل سطل آشغال می ریزیم و با خیال اسوده از این که یک باری را از دوش خود کنار زده ایم به راه مان ادامه می دهیم،خب گرگعلی پاکبان محل باید این پوست پفک را برداره و بندازه تو ماشین زباله روب و ماشین زباله روب هم باید این پوست پفک را ببره به سمت محل بازیافت و..... خب پوست پفک را می توان یک مشکل و استثنایی فرض کرد که ابتدا مربوط به ما میشد اما ما آن را در سطل زباله **پرتاب کردیم** و بعدش گرگعلی هم این پوست رو از سطل زباله به داخل ماشین زباله روب **پرتاب کرد** و ماشین زباله روب هم این پوست پفک را در محل بازیافت **پرتاب کرد** و..... پس همین جور پوست پفک دست به دست و پرتاب میشه .....تا به یک ناکجا آبادی برسه! ☺

در برنامه نویسی جاوا نقش کلمه کلیدی Throws هم همین طور است یعنی وقتی شما در متد خود از این کلمه کلیدی Throws استفاده می کنید و در صورت رخ دادن استثنا ، کلمه کلیدی Throws این استثنا را پرتاب می کند به مکانی دیگر و شما را از شر و سنگینی استثنا نجات می دهد. همین جور استثنا دست به دست می شود و.....

امیدوارم با این مثال مفهوم تا حدودی روشن شده باشد حالا میریم سراغ مثال کدنویسی آن:

مثال:

```
package javalike_Exception;

public class Test {

	public void division(int a, int b) throws ArithmeticException {

		System.out.println(a / b);

	}

}
```

- همان طور که مشاهده می کنید ما یک متد تعریف کردیم که دو پارامتر میگیرد و متغیر اول را بر متغیر دوم تقسیم می کند. از آنجایی که احتیاط شرط عقل است و امکان داره کاربر متغیر b را صفر دهد ما از مدیریت استثنا استفاده کرده ایم.
- برای مدیریت استثنا از کلمه کلیدی throws استفاده کرده ایم.
- همان طور که میبینید کلمه کلیدی throws بعد از امضا یا پرانتز جلوی متد قرار گرفته است، پس مکان قرار گیری کلمه کلیدی throws بعد از پرانتز جلوی متد می باشد.
- بعد از کلمه کلیدی throws نام کلاس استثنایی که احتمال رخ دادن انها وجود دارد را می اوریم. بعدش بلوک متد را باز و بسته کرده و دستورات خود را درون آن قرار می دهیم.
- پس متد را به روش معمول پیاده سازی می کنیم و تنها کلمه کلیدی throws همراه با نام کلاس استثنایی که احتمال رخ دادن آن وجود دارد را در کنار پرانتز جلوی متد قرار می دهیم.
- در این مثال چون محاسبات تقسیم هست از استثنایی که در مورد ریاضیات رخ می دهد یعنی ArithmeticException استفاده کرده ایم.

همچنین ما می توانیم بعد از کلمه کلیدی throws به تعداد دلخواه کلاس استثنای قرار دهیم:

```
package javalike_Exception;

public class Test {

	public void division(int a, int b) throws
ArithmeticException,NullPointerException{

		System.out.println(a / b);

	}
}
```

بعد از کلمه کلیدی throws از دو استثنای ArithmeticException,NullPointerException استفاده شده است.

نکته : متدی که در آن استثنا throws شده است هنگامی که در مکانی از برنامه آن را صدا می زنیم، باید آن متد در بلوک try-catch قرار بگیرد چون حاوی استثنا می باشد.

مثال:

```
package javalike_Exception;

public class Test {

	public void division(int a, int b) throws Exception {

		System.out.println(a / b);

	}

	public static void main(String[] args) {
		Test t = new Test();

		try {
			t.division(10, 0);
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		System.out.println(":-)");
	}

}
```

خروجی:

```
java.lang.ArithmeticException: / by zero
	at javalike_Exception.Test.division(Test.java:7)
	at javalike_Exception.Test.main(Test.java:15)
:-)
```

```
public void division(int a, int b) throws Exception {

		System.out.println(a / b);

	}
```

برای این متد ،کلاس استثنای Exception را throws کردیم.

```
t.division(10, 0);
```

وقتی خواستیم متد .division را صدا بزنیم برنامه هشدار بهمون داد که یا به متد main کلاس استثنا throws کنید یا از متد try-catch استفاده کنید. و ما هم از متد try-catch استفاده کردیم:

```
    try {
            t.division(10, 0);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

بلوک Finally :

بلوک Finally به دنبال بلوک try-catch و بعد از بلوک catch قرار می گیرد .

کدهای درون بلوک Finally صرف نظر از این که استثنایی رخ دهد یا خیر همیشه و در هر صورتی اجرا می شوند.

بلوک Finally به شما اجازه میدهد هر دستوری را اجرا کنید و اصلا هم براش مهم نیست که در بلوک try چه اتفاقی خواهد افتاد.

نحوه نوشتن :

بلوک Finally بعد از بلوک catch قرار میگیرد. شکل پیاده سازی آن بصورت زیر است:

نکته: به نحوه نوشتن یک دستوری در برنامه نویسی Syntax (سینتکس) می گویند.

Syntax

```
try {
        // Protected code
    }catch(ExceptionType1 e1) {
        // Catch block
    }catch(ExceptionType2 e2) {
        // Catch block
    }catch(ExceptionType3 e3) {
        // Catch block
    }finally {
        // The finally block always executes.
    }
```

نکته: هر بلوک try-catch تنها می تواند یک بلوک final داشته باشد.

مثال:

```
package javalike_Exception;

public class Test {

	public static void main(String[] args) {
		int a[] = new int[2];
		try {
			System.out.println("Access element three :" + a[3]);
		} catch (ArrayIndexOutOfBoundsException e) {
			System.out.println("Exception thrown  :" + e);
		} finally {
			a[0] = 6;
			System.out.println("First element value: " + a[0]);
			System.out.println("The finally statement is executed");
		}
	}
}
```

خروجی:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

```
System.out.println("Access element three :" + a[3]);
```

چون اندیس این آرایه بیشتر از اندازه آرایه است استثنا رخ می دهد و برنامه دستورات درون بلوک `catch` و نهایت دستورات درون بلوک `finally` را اجرا میکند.

برای روشن شدن این مفهوم که حتی اگر در بلوک `try` استثنایی هم رخ ندهد دستورات درون بلوک `finally` اجرا می شود به مثال زیر توجه کنید:

```
package javalike_Exception;

public class Test {

	public static void main(String[] args) {
		int a[] = new int[2];
		a[1]=500;
		try {
			System.out.println("Access element one :" + a[1]);
		} catch (ArrayIndexOutOfBoundsException e) {
			System.out.println("Exception thrown  :" + e);
```

```
            } finally {
                  a[0] = 100;
                  System.out.println("First element zero: " + a[0]);
                  System.out.println("The finally statement is executed");
            }
      }
}
```

خروجی:

```
Access element one :500
First element zero: 100
The finally statement is executed
```

در این مثال هیچ استثنایی رخ نداده و دستورات درون بلوک **try** و بلوک **finally** اجرا شده است.

به نکات زیر توجه کنید:

۱. یک بوک **catch** بدون دستور **try** نمی تواند وجود داشته باشد. یعنی این که بلوک **catch** حتما همراه آن باید بلوک **try** را پیاده سازی کنیم.

```
package javalike_Exception;

public class Test {

      public static void main(String[] args) {
            int a[] = new int[2];
            a[1]=500;
       catch (ArrayIndexOutOfBoundsException e) {
                  System.out.println("Exception thrown  :" + e);
            }
}
}
```

خروجی: خطای کامپایل!!!!!!!!!!

۲. اجباری نیست که حتما همراه با بلوک **try-catch** از بلوک **finally** استفاده کنیم.

```
package javalike_Exception;

public class Test {

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

```
            int a[] = new int[2];
            a[1]=500;
            try {
                  System.out.println("Access element one :" + a[3]);
            } catch (ArrayIndexOutOfBoundsException e) {
                  System.out.println("Exception thrown  :" + e);
            }
      }
}
```

خروجی:

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
```

۳. بلوک **try** می تواند بدون **catch** و تنها با بلوک **finally** پیاده سازی شود:

مثال:

```
package javalike_Exception;

public class Test {

      public static void main(String[] args) {
            int a[] = new int[2];
            a[1]=500;
            try {
                  System.out.println("Access element one :" + a[3]);
            } finally {
                  a[0] = 100;
                  System.out.println("First element zero: " + a[0]);
                  System.out.println("The finally statement is executed");
            }
      }
}
```

خروجی:

```
First element zero: 100
The finally statement is executed
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
      at javalike_Exception.Test.main(Test.java:9)
```

۴. بلوک **try** نمی تواند بدون بلوک **catch** یا **finally** پیاده سازی شود:

```
package javalike_Exception;

public class Test {

	public static void main(String[] args) {
		int a[] = new int[2];
		a[1]=500;
		try {
			System.out.println("Access element one :" + a[1]);
		}

	}
}
```

خروجی: خطای کامپایل!!!!

۵. هیچ کدی نمی تواند در بین بلوک های try،catch و finally قرار بگیره:

```
package javalike_Exception;

public class Test {

	public static void main(String[] args) {
		int a[] = new int[2];
		a[1]=500;
		try {
			System.out.println("Access element one :" + a[1]);
		}
		System.out.println("Error");
		catch (ArrayIndexOutOfBoundsException e) {
			System.out.println("Exception thrown  :" + e);
		}
		System.out.println("Error");
		finally {
			a[0] = 100;
			System.out.println("First element zero: " + a[0]);
			System.out.println("The finally statement is executed");
		}
	}
}
```

خروجی: خطای کامپایل، زیرا خط کدهای نارنجی رنگ بین بلوک های try،catch و finally قرار گرفته اند و طبق قانون اشتباه می باشد و هیچ کدی حق قرار گیری میان بلوک های try،catch و finally را ندارد.

ایجاد استثنا برای کاربر:

شما می توانید برای خودتون در جاوا استثنا ایجاد کنید.یعنی چی؟! یعنی برای خودتون استثنا بنویسید و استثنا تولید کنید!!! مثلا می تونید استثنایی بنویسید اگر کاربر مقدار یک متغیر را عددی منفی داد برنامه دچار استثنا شود!!! یعنی مثل استثناهای ریاضیات و اندیس آرایه و `null` بودن متغیر و... که در جاوا اماده هستند شما نیز می توانید استثناهای مورد نظر خود را بنویسید و در مواقع نیاز از آنها استفاده کنید.

روش تولید استثنا:

همان طور که میدانید هر استثنایی یک کلاس بود که از آنها استفاده می کردیدم.پس ما نیز برای تولید استثنا نیاز به تعریف کلاس به روش معمول داریم.

آموزش این قسمت بصورت مثال محور جلو میرویم:

ما قصد داریم استثنایی تولید کنیم که اگر کاربر مقداری منفی وارد کرد استثنا رخ دهد.این استثنا در موارد سن انسان، مقدار اضلاع اشکال هندسی و جاهایی که نمی توانیم مقدار منفی دهیم کاربرد دارد.

پس ما ابتدا نیاز به تعریف یک کلاس برای استثنای خود داریم:

```
package javalike_Exception;

public class NegativeNumber {


}
```

- اگر قصد دارید استثنا هنگام کامپایل رخ دهد ، کافیست که کلاس شما ، کلاس `Exception` به ارث ببرد.
- اگر قصد دارید استثنا هنگام زمان اجرا رخ دهد ، کافیست که کلاس شما ، کلاس `RuntimeException` به ارث ببرد.

در حال حاضر قصد داریم استثنا در زمان کامپایل رخ دهد پس کلاس `Exception` را به ارث می بریم:

```
package javalike_Exception;

public class NegativeNumber extends Exception {


}
```

خب شما کلاس استثنای خود را به همین راحتی ایجاد کردید!!!

الان دیگه کلاس NegativeNumber کلاس استثنایی هست که شما می توانید در مواردی که احتمال رخ دادن عدد منفی وجود دارد از آن استفاده کنید، در ضمن یک کلاس استثنا مانند سایر کلاس ها می تواند متغیر و متد و... داشته باشد.خب تمام شد؟!! خیر صبر کنید ، کار هنوز تمام نشده است ☺

ما میخوایم یک کلاس **آدم** ایجاد کنیم که دارای دو متغیر **نام و سن** می باشد. و این کلاس ادم ما دو متد setter و getter داره که سن رو مقداردهی می کنه و دیگری مقدار سن را برمیگردونه، خب برای این که امنیت برنامه را بالا ببریم باید کاری کنیم که کاربر نتونه سن منفی به سن ادم بیچاره ما بدهد، چون اگه سن منفی بده آدمک مون میمیره ☺

```
package javalike_Exception;

public class Human {

	String name;
	int age;

	public int getAge() {
		return age;
	}

	public void setAge(int age){
		this.age = age;
	}

}
```

خب ما یک کلاس ادم به نام Human ایجاد کرده که دارای دو متغیر و دو متد می باشد.تا اینجا مشکلی نیست!! اما به متد زیر توجه کنید:

```
	public void setAge(int age){
		this.age = age;
	}
```

اگه کاربر مقدار age رو منفی بده منطق برنامه اشتباه میشه چون سن هیچ انسانی منفی نیست! پس باید در این متد سازو کاری به هم بزنیم که اجازه نده که مقدار منفی به age داده بشه، برای این کار باید از کلاس استثنایی که از قبل تعریف کردیم استفاده کنیم.چگونه؟!

کافیست در بدنه متد و قبل از ست شدن متغیر age کلاس استثنای خود را ایجاد کنیم،روش ایجاد بصورت زیر است:

```
public void setAge(int age) {
		if (age < 0)
			throw new NegativeNumber();
		this.age = age;
	}
```

یک شرط گذاشتیم که اگر مقدار متغیر age کمتر از صفر یعنی منفی بود یک استثنا تولید کن ، خب چطور استثنا تولید میشه؟! کافیست ابتدا کلمه کلیدی throw بعدش کلمه کلیدی new و بعد از آن سازنده کلاس استثنایی که از قبل ایجاد کردیم را صدا بزنیم:

```
Throw + new + NegativeNumber();
```

Throw + new + سازنده کلاس استثنایی که از قبل ایجاد کردیم;

با این کار اگر شرط if برقرار باشد ،استثنای مورد نظر تولید میشود. خب همان طور که از قبل بررسی کردیم جایی که استثنا رخ میدهد را باید کنترل کرد برای کنترل کردن هم یا از بلوک try-catch یا از کلمه کلیدی Throws باید استفاده کرد. خب ما نیز کلمه کلیدی Throws همراه با نام کلاسی که استثنای آن را تولید کردیم را کنار پرانتز متد setAge قرار می دهیم :

```
	public void setAge(int age) throws NegativeNumber {
		if (age < 0)
			throw new NegativeNumber();
		this.age = age;
	}
```

حالا کلاس Human بصورت زیر تغییر یافته است:

```
package javalike_Exception;

public class Human {

	String name;
	int age;

	public int getAge() {
		return age;
	}

	public void setAge(int age) throws NegativeNumber {
		if (age < 0)
			throw new NegativeNumber();
		this.age = age;
	}

}
```

تا اینجا دیگه تونستیم کلاس استثنا با نام NegativeNumber ایجاد کنیم و بعدش از این استثنا در کلاس Human برای تولید استثنا استفاده کرده ایم.

حالا در کلاسی مانند کلاس Test این برنامه رو تست و امتحان میکنیم:

در کلاس Test یک شی با نام h از کلاس Human ایجاد کرده ایم.بعدش با شی h متد setAge را صدا زده و مقدار -5 را به آن داده ایم ، چون متد setAge به استثنای NegativeNumber ، throws شده است هنگام استفاده از آن باید از بلوک try-catch استفاده کنیم:

```
package javalike_Exception;

public class Test {

	public static void main(String[] args) {
	Human h=new Human();
	try {
		h.setAge(-5);
			} catch (NegativeNumber e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}

	}
}
```

خروجی: بعد کامپایل و اجرای این کلاس خروجی بصورت زیر است:

```
javalike_Exception.NegativeNumber
	at javalike_Exception.Human.setAge(Human.java:14)
	at javalike_Exception.Test.main(Test.java:8)
```

- همان طور که مشاهده میکنید استثنایی که رخ داده است از نوع کلاس استثنایی با نام NegativeNumber که ایجاد کرده ایم می باشد.
- دلیل رخ دادن چنین استثنایی این هست که ما مقداری منفی را به سن آدم داده ایم.
- خب امیدوارم روش تولید استثنا را یاد گرفته باشید ، اگه هم جواب منفی است اصلا نگران نباشید طبیعی است چندین بار این آموزش را مرور کنید و در کنارش تمرین و مثال حل کنید مطئنا یادگیری بهتر خواهد شد.

مثال :این کلاس استثنایی هست که قرار است تولید کنیم.

```
package javalike_Exception;
// File Name InsufficientFundsException.java
```

```
public class InsufficientFundsException extends Exception {
   private double amount;

   public InsufficientFundsException(double amount) {
      this.amount = amount;
   }

   public double getAmount() {
      return amount;
   }
}
```

کلاسی که در آن استثنا تولید کرده ایم.

```
package javalike_Exception;
// File Name CheckingAccount.java


public class CheckingAccount {
   private double balance;
   private int number;

   public CheckingAccount(int number) {
      this.number = number;
   }

   public void deposit(double amount) {
      balance += amount;
   }

   public void withdraw(double amount) throws InsufficientFundsException {
      if(amount <= balance) {
         balance -= amount;
      }else {
         double needs = amount - balance;
         throw new InsufficientFundsException(needs);
      }
   }

   public double getBalance() {
      return balance;
   }

   public int getNumber() {
      return number;
```

```
   }
}
```

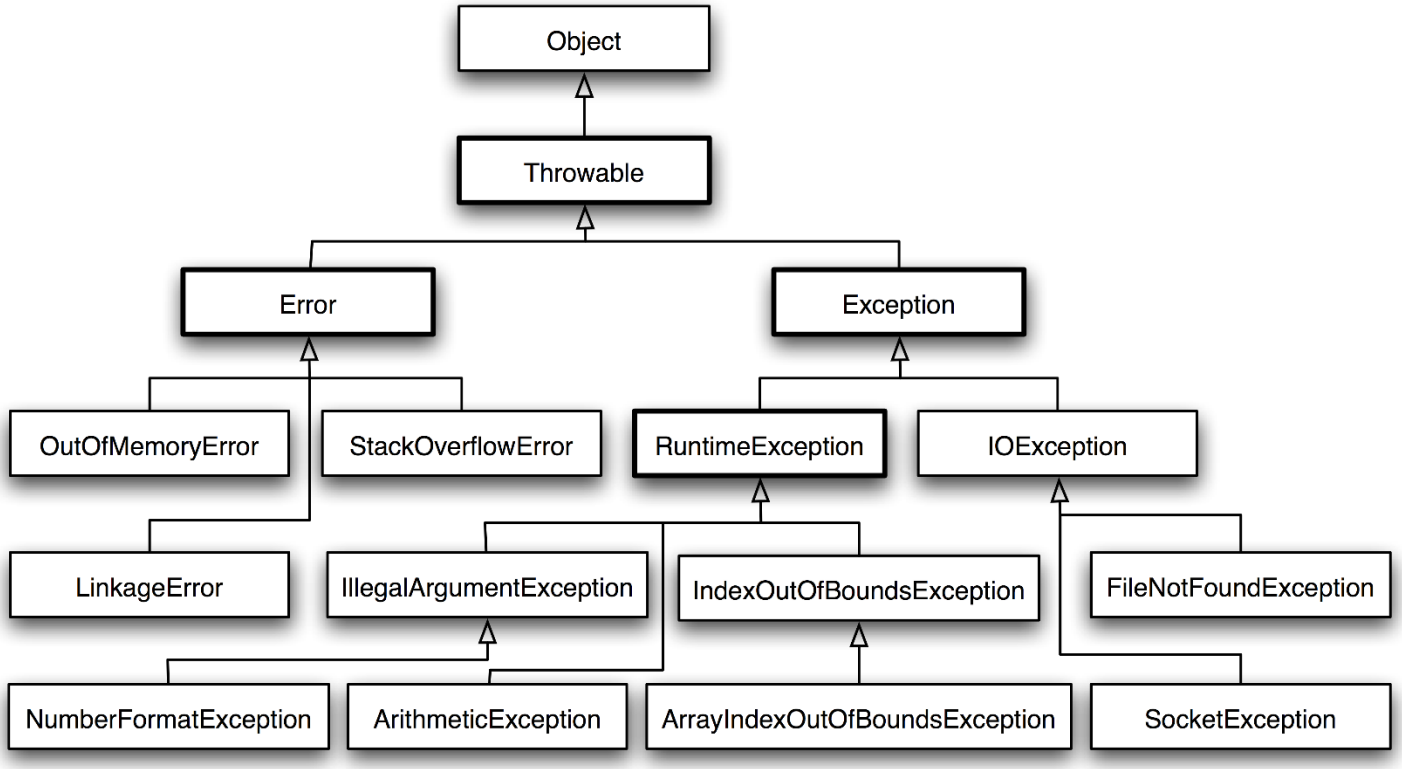
کلاس برای تست و اجرای برنامه:

```
package javalike_Exception;
// File Name BankDemo.java
public class BankDemo {

   public static void main(String [] args) {
      CheckingAccount c = new CheckingAccount(101);
      System.out.println("Depositing $500...");
      c.deposit(500.00);

      try {
         System.out.println("\nWithdrawing $100...");
         c.withdraw(100.00);
         System.out.println("\nWithdrawing $600...");
         c.withdraw(600.00);
      }catch(InsufficientFundsException e) {
         System.out.println("Sorry, but you are short $" + e.getAmount());
         e.printStackTrace();
      }
   }
}
```

خروجی:

```
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Sorry, but you are short $200.0
javalike_Exception.InsufficientFundsException
     at javalike_Exception.CheckingAccount.withdraw(CheckingAccount.java:22)
     at javalike_Exception.BankDemo.main(BankDemo.java:14)
```

سلسله مراتب استثناهای معروف را در زیر مشاهده میکنید:



تفاوت میان کلمات کلیدی Throwو Throws :

به زبان ساده و کوتاه تفاوت این دو کلمه کلیدی در این هست که کلمه کلیدی Throws برای کنترل و مدیریت استثنا استفاده میشه و کلمه کلیدی Throw برای تولید استثنا استفاده می شود.

مثال و تمرین حل کنید که این مفاهیم روشن بشه تئوری خوندن برنامه نویسی فایده نداره!!!! بعد از پایان مفاهیم تئوری ان شاالله آموزش پروژه محور از ابتدای شروع جاوا شروع میکنیم میایم به جلو لازمش هم اینه که شما این مفاهیم رو بدونید.

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.