

به نام خدا



برنامه‌سازی پیشرفته

دانشگاه شهید بهشتی . دانشکده مهندسی کامپیوتر

دکتر مجتبی وحیدی اصل

مدیریت استثنائات

سارا شیری

فهرست مطالب

1. استثنا چیست؟
2. مقدمه‌ای بر مدیریت استثنایات
3. انواع استثنا و تفاوت‌های آن
4. کلیدواژه‌های مهم در مدیریت استثنایات
5. انتشار استثنا
6. مدیریت استثنا در هنگام بازنویسی متدهای احتمالی
7. طراحی استثنا توسط برنامه‌نویس

امروز چه می‌آموزیم؟

مدیریت استثنایات یک سازوکار قدرتمند برای مدیریت (هندل کردن) خطاها زمان اجرا است تا در نتیجه این سازوکار، برنامه در مسیر نرمال به اجرای خود ادامه دهد. در این درس، درباره استثنایات جاوا، انواع آن و تفاوت میان استثنایات چک شده و چک نشده صحبت خواهیم کرد.

استثنا چیست؟

مفهوم لغوی استثنا: به شرایط غیر عادی، استثنا گفته می‌شود.

در جاوا، استثنا به رویدادی گفته می‌شود که جریان عادی برنامه را مختل می‌کند. استثنا در حقیقت شی‌ای است که در زمان اجرا پرتاب یا به اصطلاح `throw` می‌شود!

مفهوم مدیریت استثنا: مدیریت استثنا، سازوکاری برای مدیریت خطاها زمان اجرا نظیر `ClassNotFound` ، `IO` ، `SQL` ، `Remote` و ... می‌باشد.

مدیریت استثنا به ما امکان می‌دهد برنامه‌هایی بنویسیم که قادر به هندل کردن یا رفع کردن استثناهای محتمل به وقوع هستند.
بنابراین اجرای برنامه به گونه‌ای ادامه می‌یابد که انگار استثنایی به وقوع نپیوسته است.

چند مثال از استثناهای

یک استثنا ممکن است به دلایل مختلفی شامل موارد زیر اتفاق بیوافتد:

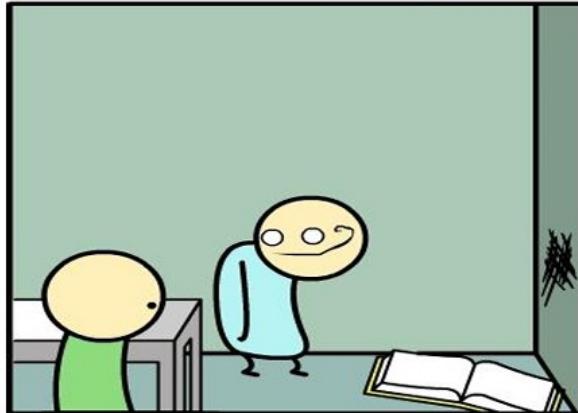
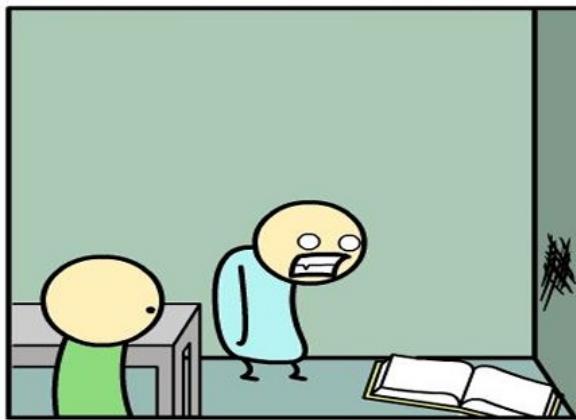
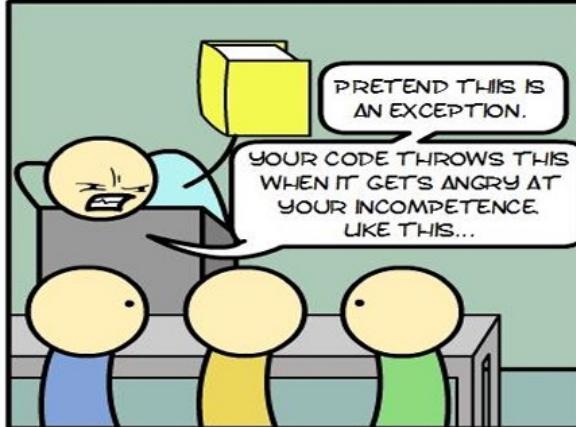
کاربر یک داده غیر مجاز وارد کند.

فایلی که می‌خواهیم باز کنیم، پیدا نشود.

اتصال یک شبکه در میانه ارتباط قطع شود.

JVM حافظه کافی نداشته باشد.

برخی از این استثنایات به سبب اشتباه کاربر رخ می‌دهند. برخی دیگر به اشتباه برنامه‌نویس مربوط می‌شوند و دلیل بروز برخی دیگر منابع فیزیکی و سیستمی هستند که به شکل‌های مختلف، دچار اشکال در عملکرد شده‌اند.



مزیت مدیریت استثنا

مزیت اصلی مدیریت استثنایات، حفظ جریان طبیعی برنامه می‌باشد. سناریوی زیر را در نظر بگیرید:

فرض کنید در برنامه، 10 دستور داشته باشیم و در دستور پنجم استثنایی رخ دهد، در نتیجه این استثنا، در بیشتر مواقع باقی کد (دستورات 6 تا 10) اجرا نخواهد شد. اگر از مدیریت استثنا استفاده کنیم، ادامه کد اجرا خواهد شد.

به همین دلیل است که از مدیریت استثنایات استفاده می‌کنیم.

statement 1;

statement 2;

statement 3;

statement 4;

statement 5; //exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

statement 10;

سوال‌هایی که ممکن است برایتان پیش بیايد!

استثناهای چک شده و چک نشده چه تفاوتی باهم دارند؟

در پس کد "int data = 50/0;" چه اتفاقی می‌افتد؟

چرا از چندین بلاک catch استفاده می‌کنیم؟

آیا ممکن است بلاک finally اجرا نشود؟

انتشار استثنا (Exception propagation) به چه معناست؟

تفاوت میان کلمه کلیدی throw و throws در چیست؟

چهار قاعده برای مدیریت استثنا در هنگام بازنویسی متدهای دارند؟

برنامه‌ای بدون مدیریت استثنایات

```
import java.util.Scanner;
public class DivideByZeroNoExceptionHandling
{
    public static int quotient( int numerator, int denominator )
    {
        return numerator / denominator; // possible division by zero
    }

    public static void main( String[] args )
    {
        Scanner scanner = new Scanner( System.in ); // scanner for input

        System.out.print( "Please enter an integer numerator: " );
        int numerator = scanner.nextInt();
        System.out.print( "Please enter an integer denominator: " );
        int denominator = scanner.nextInt();

        int result = quotient( numerator, denominator );
        System.out.printf(
            "\nResult: %d / %d = %d\n", numerator, denominator, result );
    }
}
```

برنامه‌ای با مدیریت استثنایات

```
import java.util.InputMismatchException;
import java.util.Scanner;

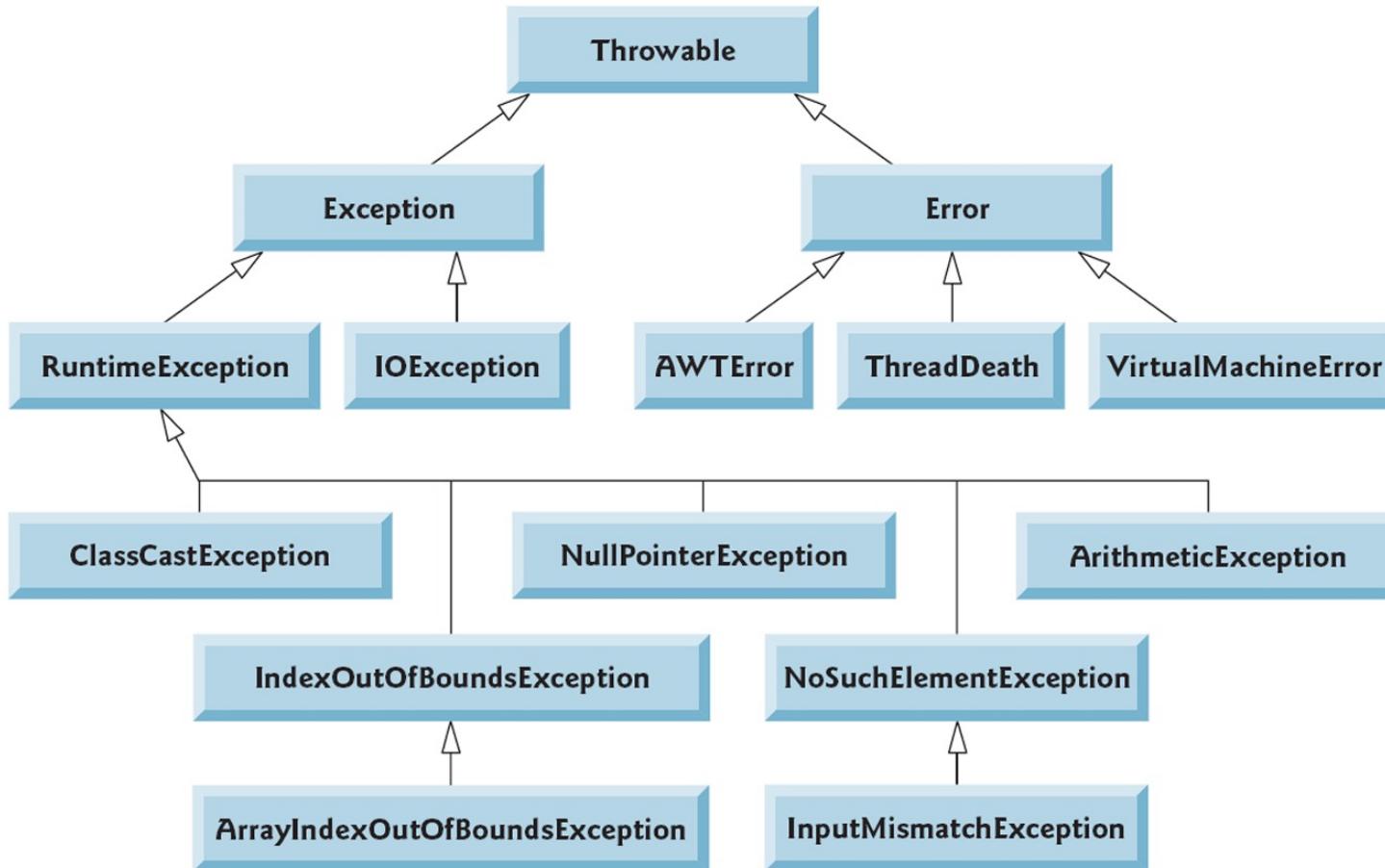
public class DivideByZeroWithExceptionHandling
{
    public static int quotient( int numerator, int denominator )
    {
        return numerator / denominator; // possible division by zero
    }

    public static void main( String[] args )
    {
        Scanner scanner = new Scanner( System.in ); // scanner for input
        boolean continueLoop = true; // determines if more input is needed
        do
        {
            try
            {
                System.out.print( "Please enter an integer numerator: " );
                int numerator = scanner.nextInt();
                System.out.print( "Please enter an integer denominator: " );
                int denominator = scanner.nextInt();

                int result = quotient( numerator, denominator );
                System.out.printf(
                    "\nResult: %d / %d = %d\n", numerator, denominator, result );
                continueLoop = false; // input successful; end looping
            } // end try
        } // end do...while
    } // end class DivideByZeroWithExceptionHandling
```

```
        int result = quotient( numerator, denominator );
        System.out.printf(
            "\nResult: %d / %d = %d\n", numerator, denominator, result );
        continueLoop = false; // input successful; end looping
    } // end try
    catch ( InputMismatchException inputMismatchException )
    {
        System.err.printf( "\nException: %s\n",
            inputMismatchException );
        scanner.nextLine(); // discard input so user can try again
        System.out.println(
            "You must enter integers. Please try again.\n" );
    } // end catch
    catch ( ArithmeticException arithmeticException )
    {
        System.err.printf( "\nException: %s\n", arithmeticException );
        System.out.println(
            "Zero is an invalid denominator. Please try again.\n" );
    } // end catch
} while ( continueLoop ); // end do...while
} // end main
} // end class DivideByZeroWithExceptionHandling
```

بخشی از ساختار سلسله مراتبی کلاس‌های Exception



متدهای Throwable

| SN | Methods with Description |
|----|--|
| 1 | public String getMessage() Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor. |
| 2 | public Throwable getCause() Returns the cause of the exception as represented by a Throwable object. |
| 3 | public String toString() Returns the name of the class concatenated with the result of getMessage() |
| 4 | public void printStackTrace() Prints the result of toString() along with the stack trace to System.err, the error output stream. |
| 5 | public StackTraceElement [] getStackTrace() Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack. |
| 6 | public Throwable fillInStackTrace() Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace. |

انواع استثنا

به طور کلی دو نوع استثنا وجود دارد. استثنای چک شده و استثنای چک نشده. به طوری که error یک استثنای چک نشده به حساب می‌آید.

طبق دسته‌بندی شرکت مایکروسیستم Sun، استثناهای سه دسته تقسیم می‌شوند:

Checked Exception

Unchecked Exception

Error

1 - استثنای چک شده: استثنایی است که عموماً نه به سبب اشتباه برنامه‌نویس بلکه به سبب مشکلی که برنامه‌نویس نمی‌توانسته آن را پیش‌بینی کند، رخداده است.

برای مثال اگر فایلی را که می‌خواهیم باز کنیم، پیدا نشود استثنای چک شده رخ می‌دهد! این نوع استثنا نمی‌تواند در زمان کامپایل مورد بی‌توجهی قرار گیرد.

کلاس‌هایی که از کلاس Throwable ارث بری می‌کنند به غیر از Runtime Exception و Error، استثناهای چک شده نامیده می‌شوند. برای مثال: IOException و غیره. استثناهای چک شده در زمان کامپایل چک می‌شوند. SQLException

2 - کلاس‌هایی که از ArithmeticException می‌کنند، استثناهای چک نشده نامیده می‌شوندو برای مثال، استثناهای NullPointerException، ArrayIndexOutOfBoundsException و غیره. این استثناهای در زمان کامپایل، در زمان اجرا چک نمی‌شوند. جلوی بروز این نوع استثنایت می‌تواند توسط برنامه‌نویس گرفته شود.

3 - خطای (error) غیر قابل ترمیم است. برای مثال .OutOfMemoryError، VirtualMachineError، AssertionError خطاهای خارج از کنترل برنامه‌نویس هستند و در زمان کامپایل بررسی نمی‌شوند.

موقعی که استثناهای چک نشده می‌تواند رخ دهد

موقعی که **ArithmeticException** رخ می‌دهد:

اگر عددی تقسیم بر صفر شود، **ArithmeticException** رخ می‌دهد.

```
int a = 50/0; //ArithmeticeException
```

موقعی که **NullPointerException** رخ می‌دهد:

هرگاه مقدار یک متغیر، null باشد، انجام هر عملیاتی توسط آن متغیر منجر به استثنای **NullPointerException** خواهد شد.

```
String s = null;  
System.out.println(s.length()); //NullPointerException
```

موقعی که **NumberFormatException** رخ می‌دهد:

فرمت نادرست هر مقدار، منجر به **NumberFormatException** می‌شود.

فرض کنید یک متغیر از نوع رشته (String) داشته باشیم که به جای کاراکترهای حرفی تشکیل شده است. تبدیل این متغیر به عدد منجر به استثنای NumberFormatException خواهد شد.

```
String s = "abc";  
int i = Integer.parseInt(s); //NumberFormatException
```

موقعی که **ArrayIndexOutOfBoundsException** رخ می‌دهد:

هرگاه مقداری را در اندیسی نادرست از یک آرایه (یا ساختار مشابه) قرار دهیم، منجر به ArrayIndexOutOfBoundsException خواهد شد.

```
int a[] = new int[5];  
a[10] = 50; //ArrayIndexOutOfBoundsException
```

واژه‌های رایج در مدیریت استثنائات

5 کلمه کلیدی که در مدیریت استثنایات استفاده می‌شوند:

try

catch

finally

throw

throws



بلاک try

کدی که ممکن است منجر به بروز (پرتاب) یک استثنای شود را در بلاک try قرار دهید. این بلاک باید در درون متد قرار گیرد و باید در ادامه آن یک بلاک catch یا finally بیاید. سایر بخش‌های کد که اطمینان داریم در آنها استثنای رخ نمی‌دهد را بعد از catch قرار می‌دهیم.

:catch با بلاک try ساختار

```
try{  
...  
}catch(Exception_class_Name refrence){}
```

:finally با بلاک try ساختار

```
try{  
...  
}finally{}
```

بلاک catch

بلاک برای هندل کردن Exception استفاده می‌شود. این بلاک باید پس از بلاک try قرار داده شود.

برنامه زیر که یک برنامه بدون مدیریت استثنایات است را ببینید:

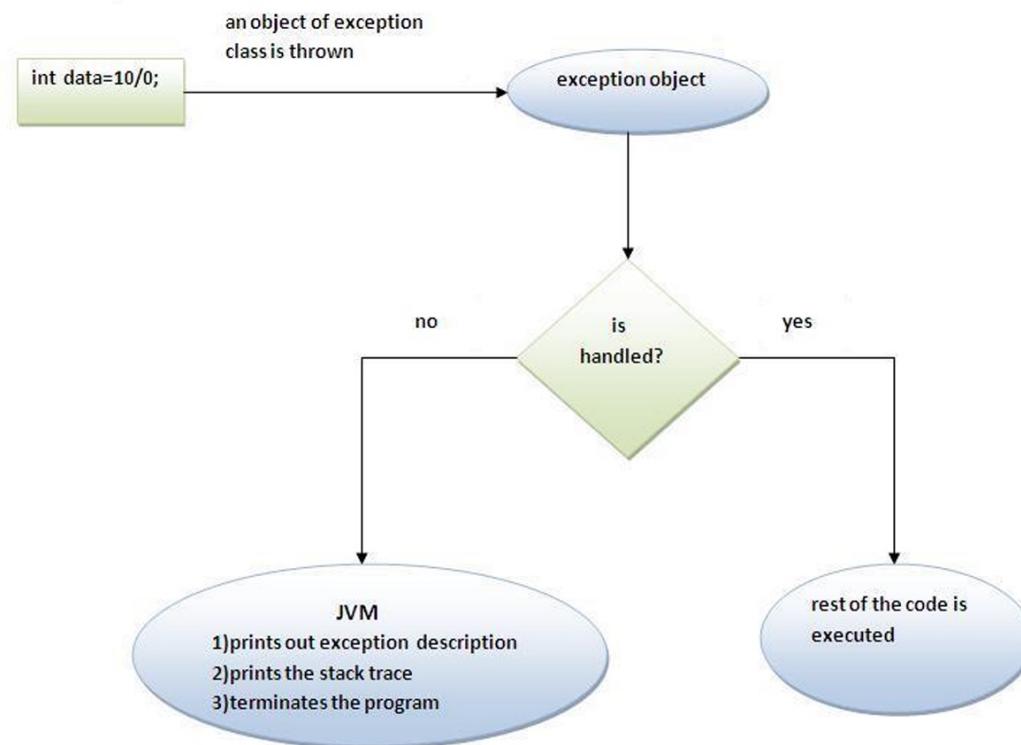
```
public class Testtrycatch1{
    public static void main(String args[]){
        int data = 50/0;

        System.out.println("rest of the code...");
    }
}
```

```
-----
java.lang.ArithmetricException: / by zero
at Testtrycatch1.main(#15:3)
at .(#16:1)
```

همانطور که می‌بینید، باقی کد یعنی چاپ رشته "the rest of code" اجرا نمی‌شود. در ادامه می‌بینیم که در پس این کد چه می‌گذرد!

پشت صحنه اجرای کد "int a = 50/0;"



JVM ابتدا بررسی می‌کند که آیا استثنا، مدیریت شده است یا خیر. اگر مدیریت نشده باشد، JVM یک اداره کننده پیش‌فرض برای استثنا در نظر می‌گیرد که کارهای زیر را انجام دهد:

توضیحی راجع به استثنا را چاپ می‌کند.

رد پشته را چاپ می‌کند (سلسله مراتبی از متدها که استثنا در آن رخ داده است).

اجرای برنامه را خاتمه می‌دهد.

اما در صورتی که استثنا توسط برنامه‌نویس مدیریت شود، جریان عادی برنامه حفظ خواهد شد و باقی‌مانده کد اجرا می‌شود.

مثال استفاده از بلاک try-catch

دقت کنید که بخشی از کد را که مطمئنیم استثنایی در آن رخ نمی‌دهد، بعد از بلاک catch بنویسیم.

```
public class Testtrycatch2{
    public static void main(String args[]){
        try{
            int data = 50/0;
        }catch(ArithmeticException e) {
            System.out.println(e);
        }

        System.out.println("rest of the code...");
    }
}
```

```
java.lang.ArithmetricException: / by zero
rest of the code...
```

همانطور که مشاهده می‌شود، باقی‌مانده کد پس از بروز استثنا اجرا خواهد شد.

مثال دیگر

در مثال زیر، آرایه‌ای به طول دو عنصر اعلان شده است. در نتیجه تلاش برای دسترسی عنصر سوم آرایه سبب پرتاپ شدن (throw) یک استثنا می‌شود.

```
import java.io.*;
Exception thrown : java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 2
Out of the block
public class ExcepTest{
    public static void main(String[] args){
        try{
            int a[] = new int[2];
            System.out.println("Access element three:" + a[3]);
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Exception thrown : " + e);
        }
        System.out.println("Out of the block");
    }
}
```

بلاک catch چندگانه

اگر بخواهیم واکنش‌های مختلف در هنگام رخداد استثناهای مختلف داشته باشیم، از بلاک `catch` چندگانه به صورت زیر استفاده می‌کنیم:

قاعده: در هر بار اجرا تنها یک استثنا رخ می‌دهد و هر بار تنها یک بلاک `catch` اجرا می‌شود.

قاعده: تمامی بلاک‌های `catch` باید از خاص‌ترین آنها به عمومی‌ترین آنها مرتب‌سازی شوند. یعنی `catch` برای `ArithmetricException` باید پیش از برای `Exception` قرار داده شود.

چون ما به `فیلد داده‌ای` دسترسی داریم که بازنویسی نشده است و در اصل امکان بازنویسی آن وجود ندارد، همیشه فقط مقدار `فیلد داده‌ای` کلاس والد قابل دسترسی می‌باشد.

```
public class TestMultipleCatchBlock {  
    public static void main(String args[]) {  
        try {  
            int a[] = new int[5];  
            a[5] = 30 / 0;  
        }  
        catch (ArithmaticException e) {  
            System.out.println("task1 is completed");  
        }  
        catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("task 2 completed");  
        }  
        catch (Exception e) {  
            System.out.println("common task completed");  
        }  
  
        System.out.println("rest of the code...");  
    }  
}
```

task1 is completed
rest of the code...

```
public class TestMultipleCatchBlock1 {
    public static void main(String args[]) {
        try {
            int a[] = new int[5];
            a[5] = 30 / 0;
        }
        catch (Exception e) {
            System.out.println("common task completed");
        }
        catch (ArithmaticException e) {
            System.out.println("task1 is completed");
        }
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("task 2 completed");
        }

        System.out.println("rest of the code...");
    }
}
```

```
catch (ArithmaticException e) {
    System.out.println("task1 is completed");
}
exception java.lang.ArithmaticException has already been caught

catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("task 2 completed");
}
exception java.lang.ArrayIndexOutOfBoundsException has already been caught
```

روند اجرایی در **catch** چندگانه

قاعده نحوی قرار دادن بلاک‌های **catch** چندگانه در برنامه به صورت زیر است:

```
try{  
    //Protected code  
}  
catch(ExceptionType1 e1)  
{  
    //Catch block  
}  
catch(ExceptionType2 e2)  
{  
    //Catch block  
}  
catch(ExceptionType3 e3)  
{  
    //Catch block  
}
```

بعد از یک بلاک try می‌تواند هر تعداد بلاک catch قرار بگیرد. اگر استثنایی در بلاک try رخ دهد، این استثنایی به اولین بلاک catch فرستاده می‌شود. اگر نوع (کلاس) شی استثنایی ارسال شده (پرتاب شده) با ExceptionType1 تطبیق داشت، در همان بلاک دریافت و هندل خواهد شد.

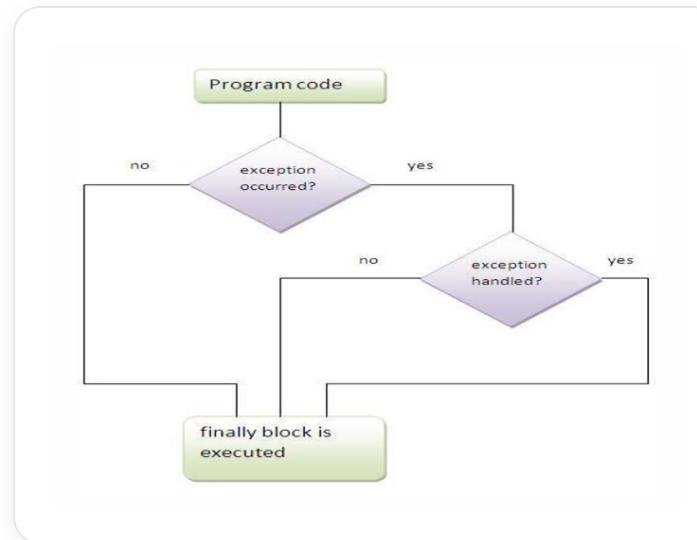
در غیر این صورت، استثنایی که در هیچ بلاک catch تطبیق پیدا نکند، در دومین دستور العمل فرستاده خواهد شد. این فرآیند تا زمانی ادامه پیدا می‌کند که یا استثنایی دریافت شود یا اینکه در هیچ بلاک catch در حالت دوم، اجرای متدهای خاتمه پیدا می‌کند و استثنایی که در هیچ بلاک catch تطبیق پیدا نکند، در دومین دستور العمل فرستاده خواهد شد.

در حالت دوم، اجرای متدهای خاتمه پیدا می‌کند و استثنایی که در هیچ بلاک catch تطبیق پیدا نکند، در دومین دستور العمل فرستاده خواهد شد.

بلاک finally

بلاک finally است که همیشه اجرا می‌شود. هدف از نوشتن این بلاک انجام برخی وظایف مهم نظیر بستن یک ارتباط شبکه‌ای، بستن یک کلاس خواندن یا نوشتن در فایل، بستن یک جریان (stream) و غیره، می‌باشد.

در واقع همه کارهایی را که می‌خواهیم پیش از توقف برنامه، حتماً انجام شوند، در بلاک finally می‌نویسیم.



قاعده: برای هر بلاک try ممکن است یک یا چند بلاک catch وجود داشته باشد، اما حداقل یک بلاک finally وجود دارد.

نکته: بلاک finally در موقع خاصی مانند فراخوانی System.exit() یا رخداد خطاهای مهلك (fatal error) که منجر به توقف پروسه اجراکننده برنامه می‌شود، اجرا نخواهد شد.

نکته: باید پس از بلاک try یا catch بیاید.

نکته: پیش از پایان برنامه، JVM یک بلاک finally را در صورت وجود، اجرا می‌کند.

سوالی که ممکن است پیش بیاید: **چرا از این بلاک استفاده می‌کنیم؟**

پاسخ این سوال این است که با این کار، یک کد پاک یا به اصطلاح cleanup مانند بستن یک فایل، بستن یک اتصال شبکه‌ای و غیره را در برنامه قرار می‌دهیم.

بلاک finally - حالت اول

حالت اول : موقعی که استثنای رخ ندهد.

```
class TestFinallyBlock {  
    public static void main(String args[]) {  
        try {  
            int data = 25 / 5;  
            System.out.println(data);  
        }  
        catch (NullPointerException e) {  
            System.out.println(e);  
        }  
        finally {  
            System.out.println("finally block is always executed");  
        }  
  
        System.out.println("rest of the code...");  
    }  
}
```

5
finally block is always executed
rest of the code...

بلاک finally - حالت دوم

حالت دوم : موقعی که استثنای خ دهد اما مدیریت نشود.

```
class TestFinallyBlock1 {  
    public static void main(String args[]) {  
        try {  
            int data = 25 / 0;  
            System.out.println(data);  
        }  
        catch (NullPointerException e) {  
            System.out.println(e);  
        }  
        finally {  
            System.out.println("finally block is always executed");  
        }  
  
        System.out.println("rest of the code...");  
    }  
}
```

5

finally block is always executed
rest of the code...

بلاک finally - حالت سوم

حالت سوم : موقعی که استثنا رخ دهد و مدیریت شود.

```
public class TestFinallyBlock2 {  
    public static void main(String args[]) {  
        try {  
            int data = 25 / 0;  
            System.out.println(data);  
        }  
        catch (ArithmaticException e) {  
            System.out.println(e);  
        }  
        finally {  
            System.out.println("finally block is always executed");  
        }  
  
        System.out.println("rest of the code...");  
    }  
}
```

```
java.lang.ArithmaticException: / by zero  
finally block is always executed  
rest of the code...
```

چند نکته

یک بلاک catch بدون وجود بلاک try معنا ندارد.

در صورت وجود بلاک‌های try/catch در برنامه اصراری بر وجود بلاک finally نمی‌باشد.

بلاک try باید با بلاک catch یا finally یا هر دو همراه باشد.

هیچ کدی نباید مابین بلاک‌های try, catch و finally قرار بگیرد.

throw کلمه

کلمه throw به این هدف استفاده می‌شود که چه در صورت وجود استثنای چک شده یا چک نشده، یک استثنای ایجاد و پرتاب شود.
اغلب در مورد استثناهای طراحی شده توسط برنامه‌نویس، کاربرد دارد.

در مثال زیر، برنامه‌نویس متوجه نوشته که یک آرگومان صحیح را دریافت می‌کند. اگر مقدار آن کمتر از 18 باشد، استثنای ArithmeticException پرتاب خواهد شد و در غیر این صورت، پیغامی مبنی بر امکان حق رای چاپ خواهد شد.

```
public class TestThrow1{
    static void validate(int age){
        if(age < 18){
            throw new ArithmeticException("not valid");
        }
        else{
            System.out.println("welcome to vote");
        }
    }

    public static void main(String[] args){
        validate(13);
        System.out.println("rest of the code...");
    }
}
```

```
-----
java.lang.ArithmetricException: not valid
        at TestThrow1.validate(#28:4)
        at TestThrow1.main(#28:12)
        at .(#29:1)
```

انتشار استثنا

استثنا در ابتدا از بالای پشته برنامه، آخرین متد فراخوانی شده، پرتاب می‌شود و اگر دریافت یا همان `catch` نشود، در پشته به سراغ متد قبلی می‌رود و اگر در آنجا هم دریافت نشود، مجدداً به سراغ متد قبل‌تر خواهد رفت و به همین ترتیب!

تا زمانی که سرانجام دریافت شود یا به پایین و انتهای پشته برسد. به این روند، انتشار استثنا گفته می‌شود.

قاعده: به طور پیش‌فرض، استثناهای چک نشده در زنجیره فراخوانی انتشار پیدا می‌کنند.

مثال از انتشار استثنا

```
class TestExceptionPropagation1{
    void m(){
        int data = 50/0;
    }

    void n(){
        m();
    }

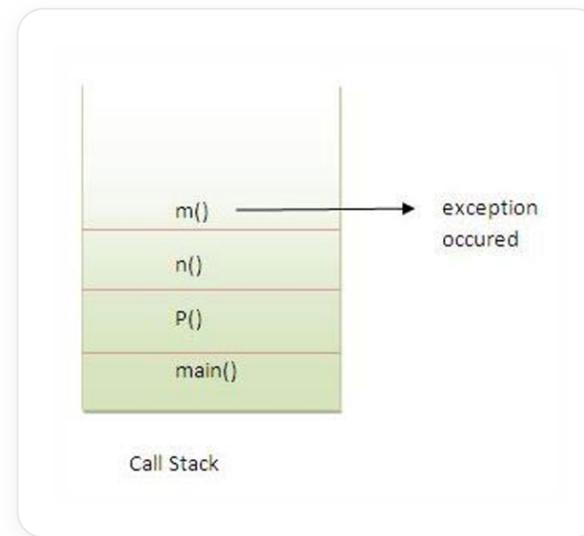
    void p(){
        try{
            n();
        } catch(Exception e) {
            System.out.println("exception handled");
        }
    }

    public static void main(String[] args){
        TestExceptionPropagation1 obj = new TestExceptionPropagation1();
        obj.p();
        System.out.println("normal flow...");
    }
}
```

exception handled
normal flow...

در مثال نشان داده شده، استثنای در متدها (m() و n()) رخ داده است و در آنجا مدیریت نشده است، در نتیجه در متدهای فراخوانده (متدهای قبلی) که (p() و p()) می‌کند که آنجا هم مدیریت نشده است. این انتشار به متدهای (p() و p()) می‌رسد که در آنجا مدیریت صورت گرفته است.

استثنای می‌تواند در هر فراخوانی متدهای پشتی فراخوانی، مدیریت شود. مثلاً در متدهای main()، p() و n()



نکته: به طور پیشفرض، استثناهای چک شده، درون پشتی فراخوانی‌ها انتشار نمی‌یابند.

در مثال زیر به عمد یک استثنای چک شده را به صورت تصنیعی ایجاد کردہ‌ایم (با استفاده از کلمه throw) و این مثال نشان می‌دهد استثناهای چک شده، انتشار پیدا نمی‌کنند.

```
class TestExceptionPropagation2{
    void m(){
        throw new java.io.IOException("device error"); //checked exception
    }

    void n(){
        m();
    }

    void p(){
        try{
            n();
        } catch(Exception e) {
            System.out.println("exception handled");
        }
    }

    public static void main(String[] args){
        TestExceptionPropagation2 obj = new TestExceptionPropagation2();
        obj.p();
        System.out.println("normal flow");
    }
}
```

Compile Time Error

throws کلمه

کلمه throws برای اعلان یک استثنا استفاده می‌شود. این کلمه اطلاعاتی به برنامه‌نویس درباره احتمال بروز یک استثنای چک شده می‌دهد و در نتیجه برنامه‌نویس را تشویق می‌کند برای مدیریت استثنا چاره‌ای بیندیشید تا جریان عادی برنامه حفظ شود.

با استفاده از اعلان استثنا (با کلمه throws) به کامپایلر می‌گوییم بررسی استثنا از زمان کامپایل به زمان اجرا منتقل شود.

سوالی که پیش می‌آید: کدام استثناهای باید اعلان شوند؟

پاسخ استثناهای چک شده است، زیرا همانطور که قبلاً دیدیم: استثنای چک نشده تحت کنترل شما است و می‌توانید کد را تصحیح کنید و همچنین درباره error مشاهده کردید که به صورت کلی تحت کنترل شما نیست. برای مثال: StackOverflowError , VirtualMachineError

قاعده نحوی throws:

```
void method_name() throws exception_class_name{  
    ...  
}
```

مزیت این کلیدواژه این است که با استفاده از آن، استثناهای چک شده نیز در زنجیره فراخوانی‌ها در پشته برنامه، انتشار می‌یابد.

```
import java.io.IOException;

class Testthrows1{
    void m()throws IOException{
        throw new IOException("device error"); //checked exception
    }

    void n(){
        m();
    }

    void p(){
        try{
            n();
        } catch(Exception e){
            System.out.println("exception handled");
        }
    }

    public static void main(String[] args){
        Testthrows1 obj = new Testthrows1();
        obj.p();
        System.out.println("normal flow...")
    }
}
```

exception handled
normal flow...

کلمه throws

نکته: اگر متدى را فراخوانی می‌کنید که ممکن است استثنایی چک شده در آن رخ دهد، دو راه دارید:

حالت اول: استثنا را با استفاده از try/catch مدیریت کنید.

حالت دوم: استثنا را مشخصا با کلمه throws اعلام کنید. به این ترتیب، استثنای احتمالی در زمان کامپایل بررسی نشده و بررسی آن به زمان اجرا موكول می‌شود.

حالت اول: استثنا مدیریت شود

اگر استثنا مدیریت شود، کد چه در صورت وجود استثنا چه در صورت عدم وجود استثنا به صورت عادی اجرا خواهد شد.

```
import java.io.*;

class M{
    void method()throws IOException{
        throw new IOException("device error");
    }
}

public class Testthrows2{
    public static void main(String[] args){
        try{
            Testthrows2 t = new Testthrows2();
            t.method();
        } catch(Exception e) {
            System.out.println("exception handled");
        }

        System.out.println("normal flow...");
    }
}
```

exception handled
normal flow...

حالت دوم: استثنا اعلان شود

در صورت اعلان استثنا، اگر استثنا رخ ندهد برنامه با موفقیت اجرا خواهد شد.

در صورت اعلان استثنا رخ دهد، استثنای مربوطه در زمان اجرا ایجاد می‌شود، زیرا کاری برای مدیریت استثنا انجام نمی‌دهد.

مثال زیر حالتی را نشان می‌دهد که استثنا رخ نداده باشد:

```
import java.io.*;

class M{
    void method() throws IOException{
        System.out.println("device operation performed");
    }
}

public class Testthrows3{
    public static void main(String[] args) throws IOException{ //declare exception

        Testthrows3 t = new Testthrows3();
        t.method();

        System.out.println("normal flow...");
    }
}
```

```
device operation performed
normal flow...
```

برنامه‌ای که در آن استثنای خ داده است:

```
import java.io.*;

class M{
    void method()throws IOException{
        throw new IOException("device error");
    }
}

public class Testthrows4{
    public static void main(String[] args)throws IOException{ //declare exception

        Testthrows4 t = new Testthrows4();
        t.method();

        System.out.println("normal flow...");
    }
}
```

Output:Runtime Exception

تفاوت میان throws و throw

| throw keyword | throws keyword |
|--|---|
| 1) throw is used to explicitly throw an exception. | throws is used to declare an exception. |
| 2) checked exception can not be propagated without throws. | checked exception can be propagated with throws. |
| 3) throw is followed by an instance. | throws is followed by class. |
| 4) throw is used within the method. | throws is used with the method signature. |
| 5) You cannot throw multiple exception | You can declare multiple exception e.g. public void method() throws IOException,SQLException. |

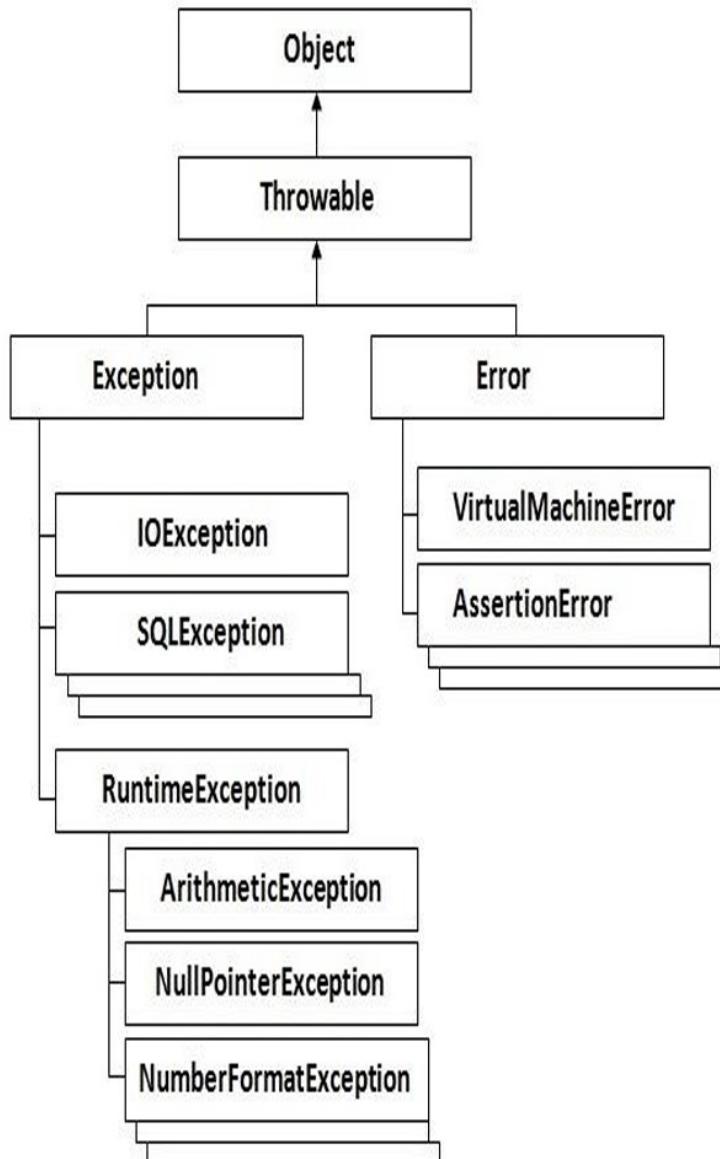
مدیریت استثنای در هنگام بازنویسی متدها

در هنگام بازنویسی متدها با استفاده از مدیریت استثنای، از قواعد مختلفی استفاده می‌شود.

این قواعد به صورت زیر می‌باشد:

اگر متدهای کلاس والد، استثنایی را اعلان نکرده باشد، متدهای بازنویسی شده کلاس فرزند می‌تواند یک استثنای چک شده را اعلان کند، اما می‌تواند استثنای چک نشده را اعلان نماید.

اگر متدهای کلاس والد، استثنایی را اعلان کرده باشد، متدهای بازنویسی شده کلاس فرزند می‌تواند همان استثنای والد را اعلان کند یا استثنایی که در سلسله مراتب استثنایها زیرکلاس است را اعلان کند یا هیچ استثنایی را اعلان نکند. اما نمی‌تواند استثنایی را اعلان کند که در درخت استثنایات پدر استثنای اعلان شده در کلاس والد بوده است.



اگر متده کلاس والد، استثنایی را اعلان نکرده باشد:

متده بازنویسی شده کلاس فرزند، نمیتواند یک استثنای چک شده را اعلان کند.

```
import java.io.*;

class Parent{
    void msg(){
        System.out.println("parent");
    }
}

class TestExceptionChild extends Parent{
    void msg()throws IOException{
        System.out.println("TestExceptionChild");
    }
}

public static void main(String[] args){
    Parent p = new TestExceptionChild();
    p.msg();
}
```

msg() in TestExceptionChild cannot override msg() in Parent
overridden method does not throw java.io.IOException

```
import java.io.*;

class Parent{
    void msg(){
        System.out.println("parent");
    }
}

class TestExceptionChild1 extends Parent{
    void msg()throws ArithmeticException{
        System.out.println("child");
    }

    public static void main(String[] args){
        Parent p = new TestExceptionChild1();
        p.msg();
    }
}
```

child

اگر متده کلاس والد، یک استثنای اعلان کند:

اگر متده کلاس والد، استثنایی را اعلان کرده باشد، متده بازنویسی شده کلاس فرزند می‌تواند همان استثنای کلاس والد یا استثنایی را اعلان کند که در درخت استثنائات، فرزند استثنای اعلان شده در کلاس والد باشد یا هیچ استثنایی را اعلان نکند. اما نمی‌تواند استثنایی را اعلان کند که در درخت استثنائات، پدر استثنای اعلان شده در کلاس والدش باشد.

در مثال زیر متده بازنویسی شده در کلاس فرزند، استثنایی را اعلان می‌کند که در درخت استثنائات، پدر استثنای اعلان شده در کلاس والدش است.

```

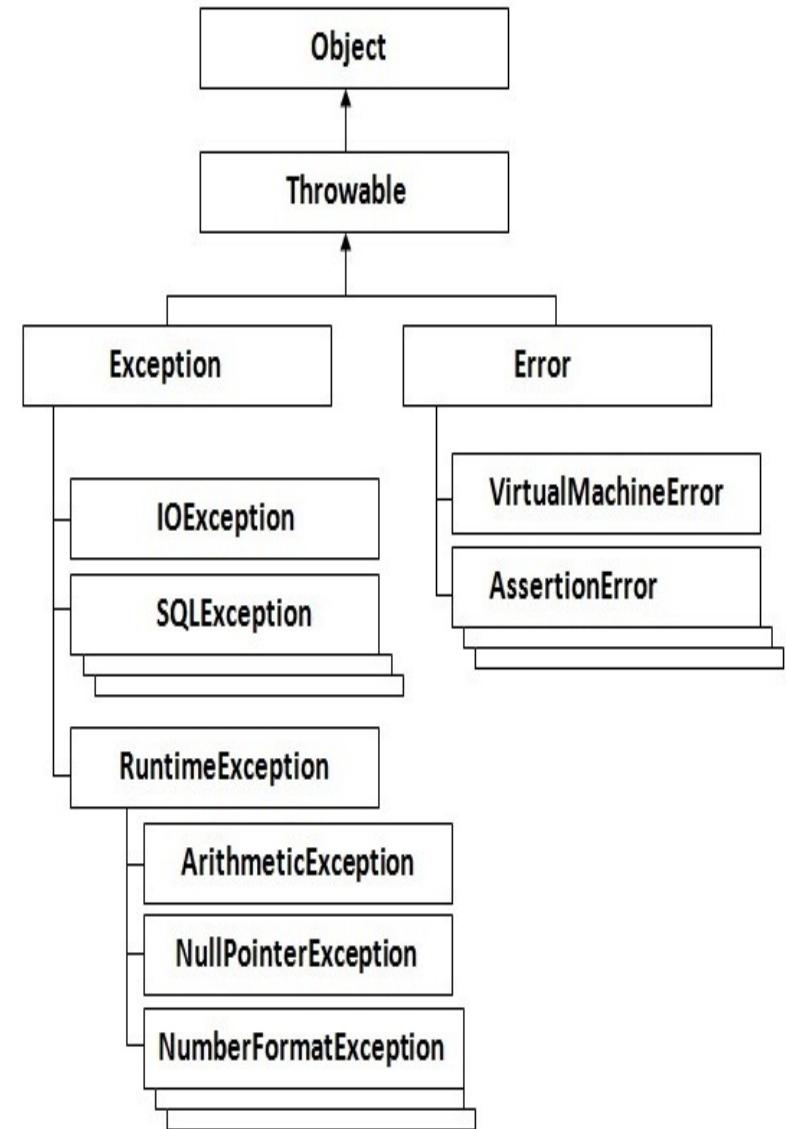
import java.io.*;

class Parent{
    void msg()throws ArithmeticException{
        System.out.println("parent");
    }
}

class TestExceptionChild2 extends Parent{
    void msg()throws Exception{
        System.out.println("child");
    }
}

public static void main(String[] args){
    Parent p = new TestExceptionChild2();
    try{
        p.msg();
    } catch(Exception e){}
}

```



msg() in **TestExceptionChild2** cannot override **msg()** in **Parent**
overridden method does not throw **java.lang.Exception**

متده بازنويسي شده کلاس فرزند، همان استثنای اعلان شده در پدر را اعلان می‌کند.

```
import java.io.*;

class Parent{
    void msg()throws Exception{
        System.out.println("parent");
    }
}

class TestExceptionChild3 extends Parent{
    void msg()throws Exception{
        System.out.println("child");
    }
}

public static void main(String[] args){
    Parent p = new TestExceptionChild3();
    try{
        p.msg();
    } catch(Exception e){}
}
```

child

```
import java.io.*;

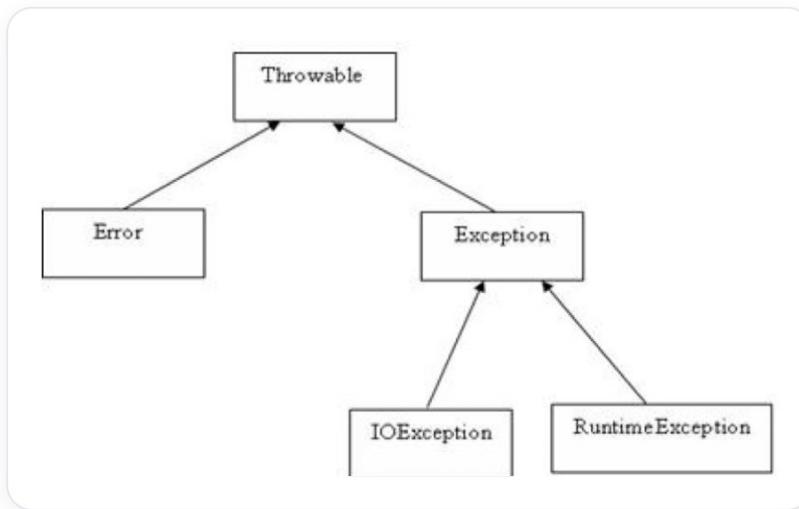
class Parent{
    void msg()throws ArithmeticException{
        System.out.println("parent");
    }
}

class TestExceptionChild4 extends Parent{
    void msg()throws ArithmeticException{
        System.out.println("child");
    }
}

public static void main(String[] args){
    Parent p = new TestExceptionChild4();
    try{
        p.msg();
    } catch(Exception e){}
}
```

child

استثناهای طراحی شده توسط برنامه نویس



استثناهای تعریف شده باید از `Throwable` ارث بری کنند.

اگر می خواهید استثنای چک شده‌ای بنویسید که به طور خودکار با یک ساختار مدیریت استثنای کنترل می شود یا در ابتدای متدهای اعلان می شود، باید از کلاس `exception` ارث بری کنید.

اگر می خواهید استثنای زمان اجرا (چک نشده) بنویسید باید از کلاس `RuntimeException` ارث بری کنید.

کلاس `Exception` خود را می توانید به صورت زیر تعریف کنید:

```
class MyException extends Exception{  
}
```

مثالی از استثنای تعریف شده توسط برنامه نویس

هرگاه برنامه نویس استثنای را در ارتباط با برنامه خود طراحی و تعریف کند:

```
class InvalidAgeException extends Exception{
    InvalidAgeException(String s){
        super(s);
    }
}

class TestCustomException1{
    static void validate(int age) throws InvalidAgeException{
        if(age < 18) {
            throw new InvalidAgeException("not valid");
        }
        else {
            System.out.println("welcome to vote");
        }
    }

    public static void main(String[] args){
        try{
            validate(13);
        } catch(Exception m){
            System.out.println("Exception occurred: " + m);
        }

        System.out.println("rest of the code...");
    }
}
```

```
Exception occurred: REPL.$JShell$19$InvalidAgeException: not valid
rest of the code...
```

مثال دیگر از استثنای تعریف شده توسط برنامه نویس

کلاس هایی که از Exception ارث بری کرده باشند، به طور پیش فرض چک شده به حساب می آیند.

برای مثال کلاس InsufficientFundsException زیر یک استثنای تعریف شده توسط برنامه نویس می باشد.

یک کلاس Exception نظیر هر کلاس دیگری در جاوا می باشد که حاوی فیلدهای داده ای و متدهای موردنیاز است.

کلاس CheckingAccount دارای متدهای withdraw() است که شی ای از کلاس استثنای InsufficientFundsException به نام () پرتاب می کند.

```
import java.io.*;

public class InsufficientFundsException extends Exception{
    private double amount;

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

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

```
import java.io.*;

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;}
}
```

```
import java.io.*;

public class BankDemo{

    public static void main(String[] args){

        CheckingAccount c = new CheckingAccount(101); //account number : 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();
        }
    }
}
```

```
Depositing $500...

Withdrawning $100...

Withdrawning $600...
Sorry, but you are short $200.0
REPL.$JShell$26$InsufficientFundsException
    at REPL.$JShell$23$CheckingAccount.withdraw($JShell$23.java:29)
    at REPL.$JShell$24B$BankDemo.main($JShell$24B.java:30)
    at REPL.$JShell$27.do_it $($JShell$27.java:18)
    at java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)
    at java.base/java.lang.reflect.Method.invoke(Method.java:578)
    at org.dflib.jjava.execution.JJavaExecutionControl.lambda$execute$1(JJavaExecutionControl.java:107)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:317)
    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1144)
    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:642)
    at java.base/java.lang.Thread.run(Thread.java:1589)
```

لیستی از استثناهای چک نشده

| Exception | Description |
|---------------------------------|---|
| ArithmaticException | Arithmatic error, such as divide-by-zero. |
| ArrayIndexOutOfBoundsException | Array index is out-of-bounds. |
| ArrayStoreException | Assignment to an array element of an incompatible type. |
| ClassCastException | Invalid cast. |
| IllegalArgumentException | Illegal argument used to invoke a method. |
| IllegalMonitorStateException | Illegal monitor operation, such as waiting on an unlocked thread. |
| IllegalStateException | Environment or application is in incorrect state. |
| IllegalThreadStateException | Requested operation not compatible with current thread state. |
| IndexOutOfBoundsException | Some type of index is out-of-bounds. |
| NegativeArraySizeException | Array created with a negative size. |
| NullPointerException | Invalid use of a null reference. |
| NumberFormatException | Invalid conversion of a string to a numeric format. |
| SecurityException | Attempt to violate security. |
| StringIndexOutOfBoundsException | Attempt to index outside the bounds of a |

لیستی از استثناهای چک شده

| Exception | Description |
|----------------------------|---|
| ClassNotFoundException | Class not found. |
| CloneNotSupportedException | Attempt to clone an object that does not implement the Cloneable interface. |
| IllegalAccessException | Access to a class is denied. |
| InstantiationException | Attempt to create an object of an abstract class or interface. |
| InterruptedException | One thread has been interrupted by another thread. |
| NoSuchFieldException | A requested field does not exist. |
| NoSuchMethodException | A requested method does not exist. |

خودمون رو بسنجیم

این بخش برای این طراحی شده که در پایان مطالعه این اسلاید، بتونی خودت رو محک بزنی و ببینی آیا مفاهیم رو به خوبی یاد گرفتی یا نه. سوالات زیر رو مرور کن و سعی کن بدون نگاه کردن به متن درس، به اون ها پاسخ بدی.

- تفاوت‌هایی که بین کلیدواژه‌های `throw` و `throws` وجود داره چه مواردی هستن؟
- بین استثناهای مختلف، کدام حالت‌شون باید اعلان بشه؟
- مهم‌ترین مزیت مدیریت استثنایات چیه؟

پایان

در صورت هرگونه سوال یا پیشنهاد میتوانید با من
در ارتباط باشید:)

gmail: sarashiri0906@gmail.com