دانشکده مهندسی و علوم کامپیوتر

برنامه نویسی پیشرفته

وحیدی اصل

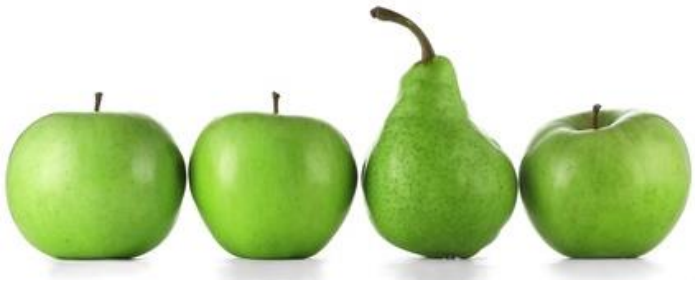
مدیریت استثنائات

امروز چه می آموزیم؟

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

استثنا چیست؟

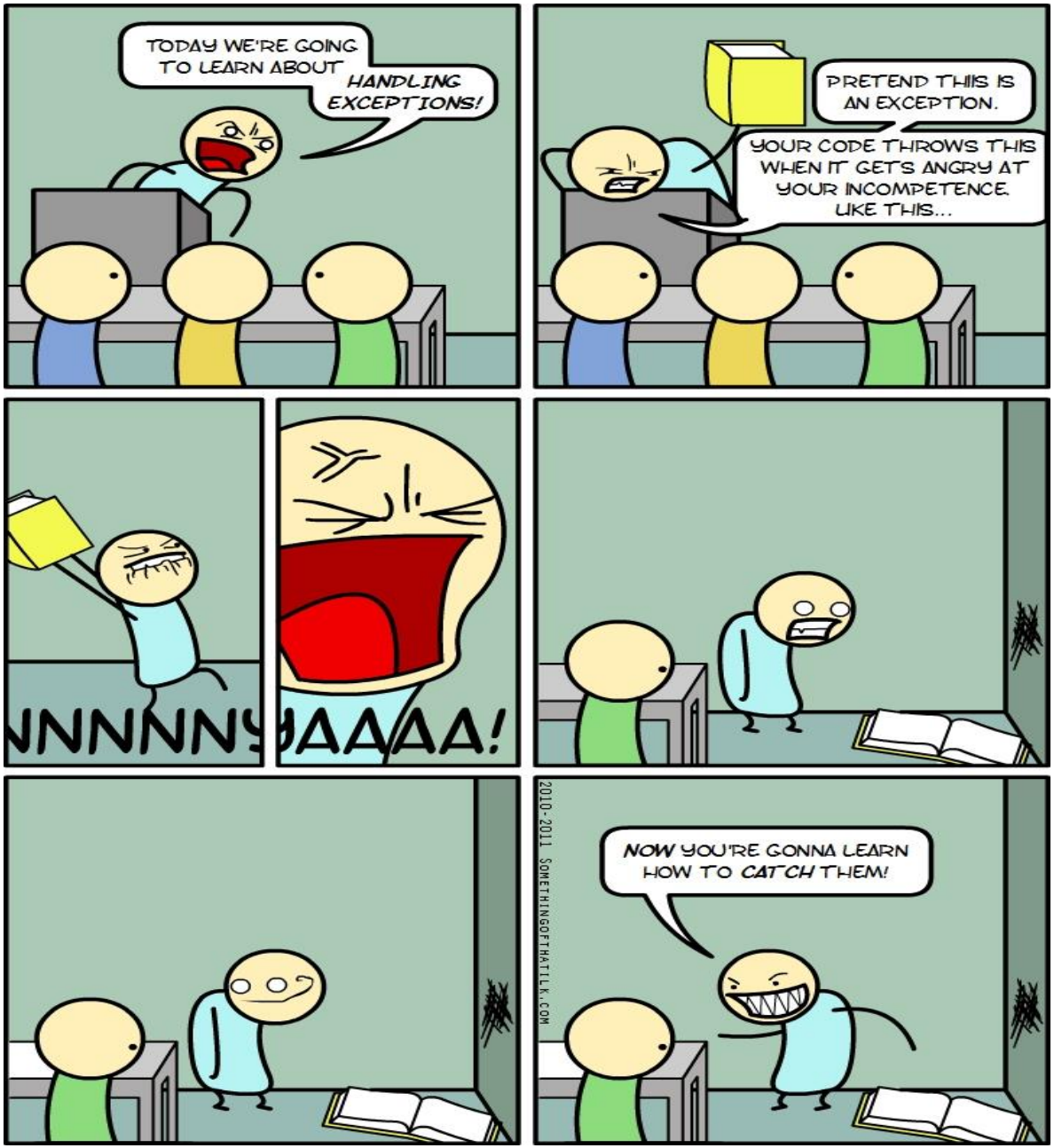
- مفهوم لغوی استثنا: به شرایط غیرعادی، استثنا گفته می شود.
- در جاوا، استثنا به رویدادی گفته می شود که جریان عادی برنامه را مختل می کند. استثنا در حقیقت شیئی است که در زمان اجرا پرتاب (throw) می شود!
- مفهوم مدیریت استثنا: مدیریت استثنا، سازوکاری برای مدیریت خطاهای زمان اجرا نظیر ClassNotFound، IO، SQL ، Remote و غیره می باشد.
- مدیریت استثنا به ما امکان می دهد برنامه هایی بنویسیم که قادرند استثناهای محتمل به وقوع را رفع (یا هندل) کنند.
  - بنابراین اجرای برنامه به گونه ای ادامه می یابد که انگار استثنایی به وقوع نپیوسته است.



چند مثال از استثناها

- یک استثنا ممکن است به دلایل مختلفی شامل موارد زیر اتفاق بیفتد:
  - کاربر یک داده غیرمجاز وارد کند.
  - فایلی را که می خواهیم باز کنیم، پیدا نشود.
  - اتصال یک شبکه در میانه ارتباط قطع شود.
  - JVM حافظه کافی نداشته باشد.
- برخی از این استثنائات به سبب اشتباه کاربر رخ می دهند، برخی دیگر به اشتباه برنامه نویس مربوط می شوند و دلیل بروز بعضی دیگر منابع فیزیکی و سیستمی هستند که به شکلهای مختلف، دچار اشکال در عملکرد شده اند.

پرتاب استثنا



مزیت مدیریت استثنائات

- مزیت اصلی مدیریت استثنائات، حفظ جریان طبیعی برنامه می باشد. سناریوی زیر را در نظر بگیرید:
- فرض کنید در برنامه، ۱۰ دستور داشته باشیم و در دستور پنجم استثنایی رخ دهد، در نتیجه این استثنا، در بیشتر مواقع باقی کد (دستورات ۶ تا ۱۰) اجرا نخواهند شد. اگر از مدیریت استثنا استفاده کنیم، ادامه کد اجرا خواهد شد.
- به همین دلیل است که از مدیریت استثنائات استفاده می کنیم.

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

آیا می دانید؟

- استثناهای چک شده و چک نشده چه تفاوتی باهم دارند؟
- در پَس کد “int data=50/0;“چه اتفاقی می افتد؟
- چرا از چندین بلاک catch استفاده می کنیم؟
- 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 متد کدامند؟

برنامه ای بدون مدیریت استثنائات

```
1  // Fig. 11.1: DivideByZeroNoExceptionHandling.java
2  // Integer division without exception handling.
3  import java.util.Scanner;
4
5  public class DivideByZeroNoExceptionHandling
6  {
7     // demonstrates throwing an exception when a divide-by-zero occurs
8     public static int quotient( int numerator, int denominator )
9     {
10       return numerator / denominator; // possible division by zero
11    } // end method quotient
12
13    public static void main( String[] args )
14    {
15       Scanner scanner = new Scanner( System.in ); // scanner for input
16
17       System.out.print( "Please enter an integer numerator: " );
18       int numerator = scanner.nextInt();
19       System.out.print( "Please enter an integer denominator: " );
20       int denominator = scanner.nextInt();
21
22       int result = quotient( numerator, denominator );
23       System.out.printf(
24          "\nResult: %d / %d = %d\n", numerator, denominator, result );
25    } // end main
26 } // end class DivideByZeroNoExceptionHandling
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at DivideByZeroNoExceptionHandling.main(
            DivideByZeroNoExceptionHandling.java:20)
```

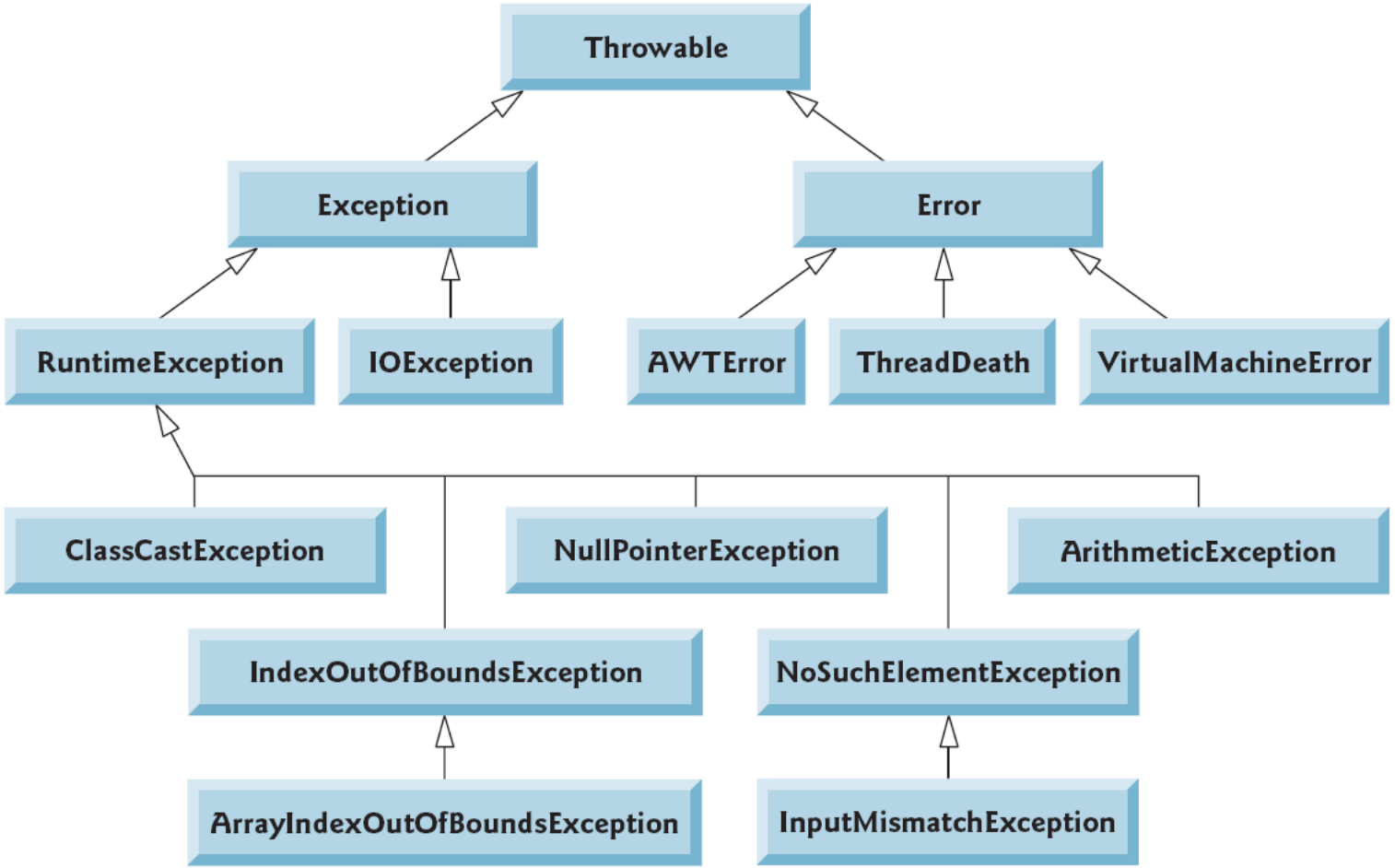
برنامه ای با مدیریت استثنائات

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

public class DivideByZeroWithExceptionHandling
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello

Exception: java.util.InputMismatchException
You must enter integers. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
      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
   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 ساختارسلسله مراتبی کلاسهای exception



Throwable متدهای

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 می آید.
- طبق دسته بندی شرکت مایکروسیستم Sun ، استثناها به سه دسته تقسیم می شوند:

- Checked Exception
- Unchecked Exception
- Error

تفاوت میان استثناهای چک شده و چک نشده

1) استثنای چک شده: استثنایی است که عموماً نه به سبب اشتباه برنامه نویس بلکه به سبب مشکلی که برنامه نویس نمی توانسته آن را پیش بینی کند، رخ داده است.
   - برای مثال اگر فایلی را که می خواهیم باز کنیم، پیدا نشود استثنای چک شده رخ می دهد! این نوع استثنا نمی تواند در زمان کامپایل مورد بی توجهی قرار گیرد.

- کلاسهایی که از کلاس Throwable ارثبری می کنند به غیر از RuntimeException و Error، استثناهای چک شده نامیده می شوند. برای مثال: IOException، SQLException و غیره. استثناهای چک شده در زمان کامپایل چک می شوند.

تفاوت میان استثناهای چک شده و چک نشده

۲) کلاسهایی که از RuntimeException ارثبری می کنند، استثناهای چک نشده نامیده می شوند. برای مثال ArithmeticException، NullPointerException، ArrayIndexOutOfBoundsException و غیره. این استثناها به جای بررسی در زمان کامپایل، در زمان اجرا چک می شوند.

—جلوی بروز این نوع از استثنئات می تواند توسط برنامه نویس گرفته شود.

۳) خطا (error) غیرقابل ترمیم است. برای مثال OutOfMemoryError، VirtualMachineError، AssertionError

- **خطاها خارج از کنترل برنامه نویس هستند و در زمان کامپایل بررسی نمی شوند.**

مواقعی که استثناهای چک نشده می توانند رخ دهند

- مواقعی که ArithmeticException رخ می دهد:
  - اگر عددی بر صفر تقسیم شود، ArithmeticException رخ می دهد.

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

- مواقعی که NullPointerException رخ می دهد:
  - هرگاه مقدار یک متغیر، null باشد، انجام هرعملیاتی توسط آن متغیر منجر به استثنای NullPointerException خواهد شد:

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

مواقعی که استثناهای چک نشده می توانند رخ دهند

- مواقعی که NumberFormatException رخ می دهد:
  - فرمت نادرست هر مقدار، منجر به NumberFormatException می شود.
  - فرض کنید یک متغیر از نوع رشته (String) داشته باشیم که به جای کارکترهای رقمی از کارکترهای حرفی تشکیل شده است. تبدیل این متغیر به عدد منجر به استثنای NumberFormatException خواهد شد.

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

- مواقعی که ArrayIndexOutOfBoundsException رخ می دهد:
- 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
```

واژه های رایج در مدیریت استثنائات

- **۵ کلمه کلیدی که در مدیریت استثنائات استفاده می شوند:**

- try
- catch
- finally
- throw
- throws

بلاک try

- کدی که ممکن است منجر به بروز (پرتاب) یک استثنا شود را در بلاک try قرار دهید. این بلاک باید درون متد قرار گیرد و باید در ادامه آن یک بلاک catch یا finally بیاید. سایر بخشهای کد که اطمینان داریم در آنها استثنا رخ نمی دهد، بعد از catch قرار می دهیم.
- ساختار try با بلاک catch:

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

- ساختار try با بلاک finally:

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

بلاک catch

- بلاک Catch برای هندل کردن Exception استفاده می شود. این بلاک باید پس از بلاک try قرار داده شود.
- برنامه بدون مدیریت استثنائات:

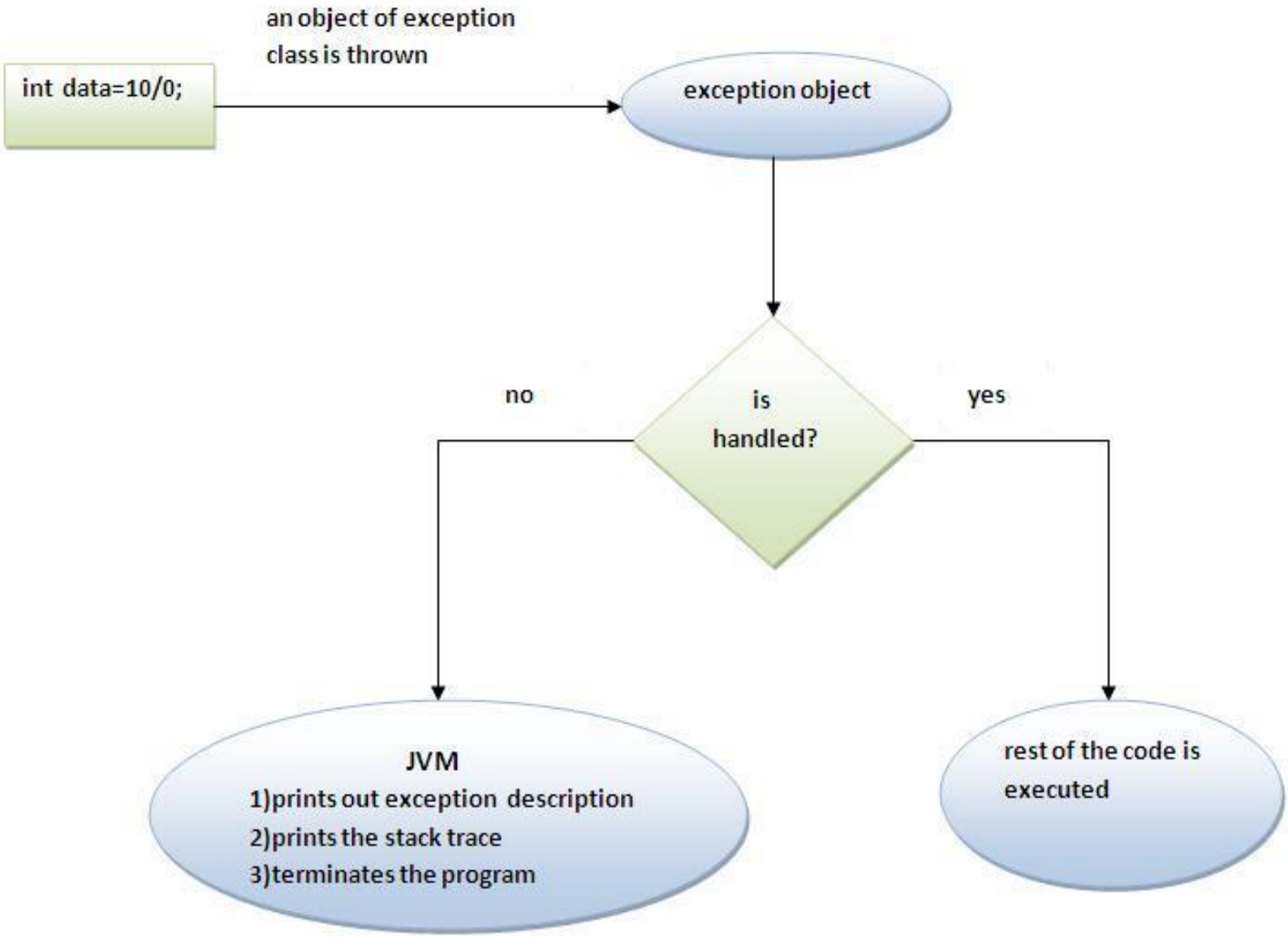
```
public class Testtrycatch1{
  public static void main(String args[]){
     int data=50/0;

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

Test it Now

```
Output:Exception in thread main java.lang.ArithmeticException:/ by zero
```

- همانطور که می بینید، باقی کد یعنی چاپ رشته "the rest of code" اجرا نمی شود. در اسلاید بعد می بینیم که در پس این کد چه می گذرد!

پشت صحنه اجرای کد "int a=50/0;"



بلاک catch

- JVM ابتدا بررسی می کند که آیا استثنا، مدیریت شده است یا خیر. اگر مدیریت نشده باشد، JVM یک اداره کننده پیش فرض برای استثنا در نظر می گیرد که کارهای زیر را انجام دهد:
  - توضیحی راجع به استثنا را چاپ می کند.
  - رد پشته را چاپ می کند (سلسله مراتبی از متدها که استثنا در آن رخ داده است).
  - اجرای برنامه را خاتمه می دهد.
  - اما در صورتی که استثنا، توسط برنامه نویس برنامه مدیریت شود، جریان عادی برنامه حفظ خواهد شد و باقیمانده کد اجرا می شود.

مثال استفاده از بلاک try-catch

```
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...");
}
}
```

Test it Now

```
Output:Exception in thread main java.lang.ArithmeticException:/ by zero
       rest of the code...
```

- دقت کنید که بخشی از کد را که مطمئنیم استثنایی در آن رخ نمی دهد، بعد از بلاک catch بنویسیم.
- همانطور که مشاهده می شود، باقیمانده کد پس از بروز استثنا اجرا خواهد شد.

مثال

- در مثال زیر، آرایه ای به طول دو عنصر اعلان شده است. در نتیجه تلاش برای دسترسی عنصر سوم آرایه سبب پرتاب شدن (throw) یک استثنا می شود.

```
// File Name : ExcepTest.java
import java.io.*;
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");
  }
}
```

خروجی برنامه

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

بلاک catch چندگانه

- اگر بخواهیم واکنشهای مختلفی در هنگام رخداد استثناهای مختلف داشته باشیم، از بلاک catch چندگانه به صورت زیر استفاده می کنیم:
- قاعده: در هر بار اجرا تنها یک استثنا رخ می دهد و هربار تنها یک بلاک catch اجرا می شود.
- قاعده: تمامی بلاکهای catch باید از به ترتیب خاصترین آنها به عمومی ترین آنها مرتب سازی شوند. یعنی catch برای ArithmeticException باید پیش از catch برای Exception قرار داده شود.

```
public class TestMultipleCatchBlock{
  public static void main(String args[]){
   try{
    int a[]=new int[5];
    a[5]=30/0;
   }
   catch(ArithmeticException 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...");
  }
}
```

Test it Now

```
Output:task1 completed
       rest of the code...
```

روند اجرایی در catch چندگانه

- قاعده نحوی قراردادن 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 می تواند هرتعداد بلاک catch قرار بگیرد. اگر استثنایی در بلاک try رخ دهد، این استثنا به اولین بلاک catch فرستاده می شود. اگر نوع (کلاس) شیئ استثنای ارسال شده (پرتاب شده) با ExceptionType1 تطبیق داشت، در همان بلاک دریافت و هندل خواهد شد.
- درغیراینصورت، استثنا به دومین دستورالعمل فرستاده خواهد شد. این فرآیند تا زمانی ادامه پیدا می کند که یا استثنا دریافت شود یا اینکه در هیچ بلاک catch تطبیق پیدا نکند.
- در حالت دوم، اجرای متد جاری خاتمه پیدا می کند و استثنا به متد قبلی درون پشته فراخوانی ها منتقل خواهد شد.

بلاک catch چندگانه

- **قاعده:** تمامی بلاکهای catch باید از به ترتیب خاصترین آنها به عمومی ترین آنها مرتب سازی شوند. یعنی برای catch ArithmeticException باید پیش از catch برای Exception قرار داده شود.

```
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(ArithmeticException e){System.out.println("task1 is completed");}
   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}

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

Test it Now

```
Output:Compile-time error
```

برنامه ای بدون مدیریت استثنائات

```
1  // Fig. 11.1: DivideByZeroNoExceptionHandling.java
2  // Integer division without exception handling.
3  import java.util.Scanner;
4
5  public class DivideByZeroNoExceptionHandling
6  {
7     // demonstrates throwing an exception when a divide-by-zero occurs
8     public static int quotient( int numerator, int denominator )
9     {
10       return numerator / denominator; // possible division by zero
11    } // end method quotient
12
13    public static void main( String[] args )
14    {
15       Scanner scanner = new Scanner( System.in ); // scanner for input
16
17       System.out.print( "Please enter an integer numerator: " );
18       int numerator = scanner.nextInt();
19       System.out.print( "Please enter an integer denominator: " );
20       int denominator = scanner.nextInt();
21
22       int result = quotient( numerator, denominator );
23       System.out.printf(
24          "\nResult: %d / %d = %d\n", numerator, denominator, result );
25    } // end main
26 } // end class DivideByZeroNoExceptionHandling
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at DivideByZeroNoExceptionHandling.main(
            DivideByZeroNoExceptionHandling.java:20)
```

برنامه ای با مدیریت استثنائات

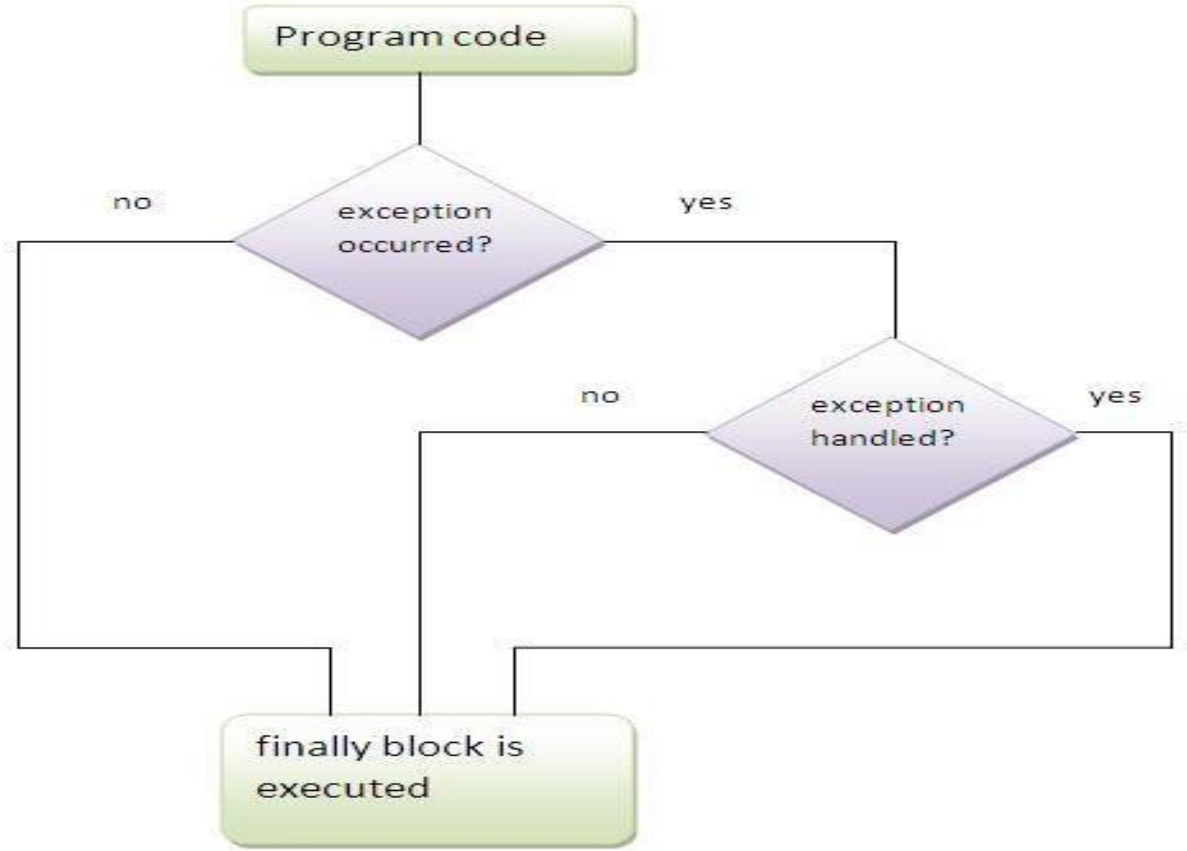
```
import java.util.InputMismatchException;
import java.util.Scanner;

public class DivideByZeroWithExceptionHandling
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello

Exception: java.util.InputMismatchException
You must enter integers. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
            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
         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
```

بلاک finally

- بلاک **finally** بلاکی است که همیشه اجرا می شود. هدف از نوشتن این بلاک انجام برخی وظایف مهم نظیر بستن یک ارتباط شبکه ای، بستن یک کلاس خواندن یا نوشتن در فایل، بستن یک جریان (stream)و غیره، می باشد.
- در واقع همه کارهایی را که می خواهیم پیش از توقف برنامه، حتماً انجام شوند، در بلاک **finally** می نویسیم.



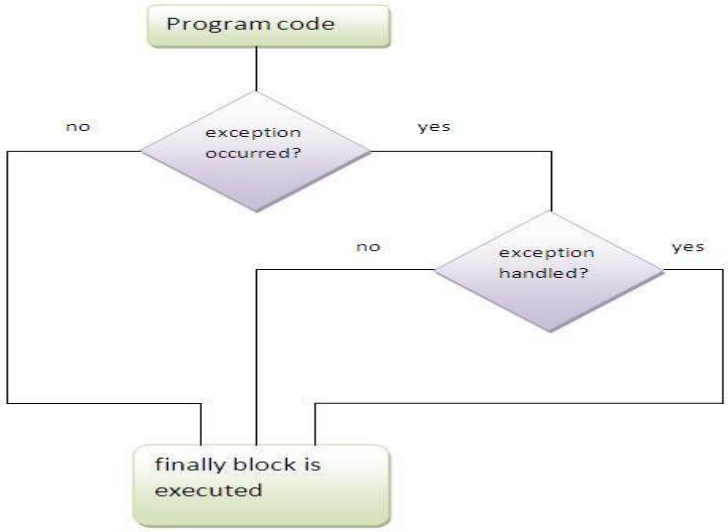
بلاک finally–حالت اول

- موقعی که استثنایی 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...");
  }
}
```

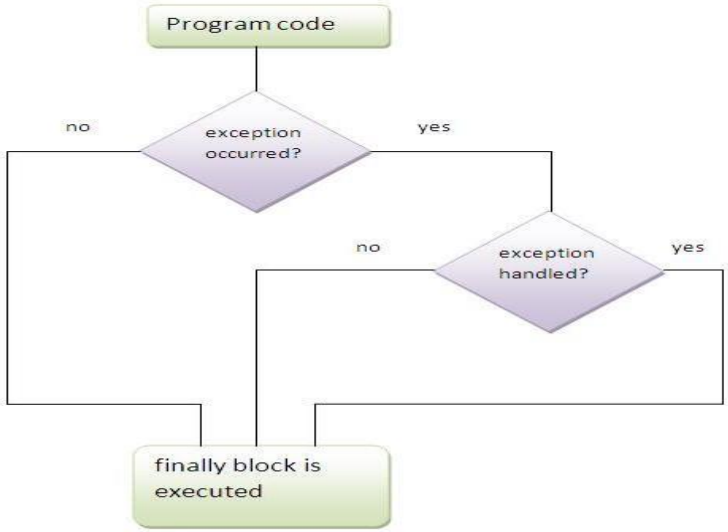
Test it Now

```
Output:5
       finally block is always executed
       rest of the code...
```



بلاک finally–حالت دوم

- موقعی که استثنایی رخ 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...");
  }
}
```

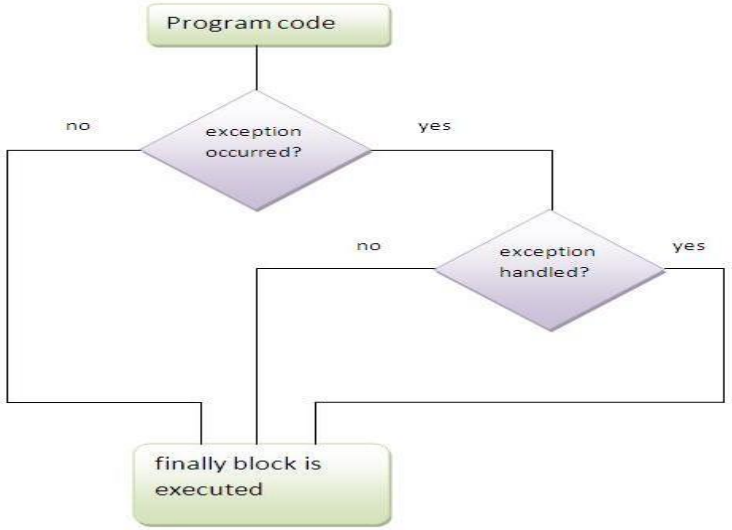
Test it Now

```
Output:finally block is always executed
       Exception in thread main java.lang.ArithmeticException:/ by zero
```

بلاک finally–حالت سوم

- موقعی که استثنایی رخ دهد و مدیریت شود.

```
public class TestFinallyBlock2{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){System.out.println(e);}

  finally{System.out.println("finally block is always executed"

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



Test it Now

```
Output:Exception in thread main java.lang.ArithmeticException:/ by zero
       finally block is always executed
       rest of the code...
```

بلاک finally

- قاعده: برای هر بلاک try ممکن است یک یا چند بلاک catch وجود داشته باشد، اما حداکثر یک بلاک finally وجود دارد.
- نکته: بلاک finally در مواقع خاصی مانند فراخوانی System.exit() یا رخداد خطاهای مهلک (fatal error) که منجر به توقف پروسس اجرا کننده برنامه می شود، اجرا نخواهد شد.

- نکته: پیش از پایان برنامه، JVM، بلاک finally را در صورت وجود، اجرا می کند.
- نکته: finally باید پس از بلاک try یا catch بیاید.
- چرا از بلاک finally استفاده می کنیم؟
  - با این کار، یک کد پاک "cleanup" مانند بستن یک فایل، بستن یک اتصال شبکه ای و غیره را در برنامه قرار می دهیم.

چند نکته

- یک بلاک catch بدون وجود بلاک try معنا ندارد.
- درصورت وجود بلاکهای try/catch در برنامه اصراری بر وجود بلاک finally نمی باشد.
- بلاک try باید با بلاک finally یا catch یا هردو همراه باشد.
- هیچ کدی نباید مابین بلاکهای try، catch و finally قرار بگیرد.

کلمه throw

- کلمه throw به این هدف استفاده می شود که چه در صورت وجود استثنای چک شده یا چک نشده، یک استثنا ایجاد (پرتاب) شود.
- اغلب در مورد استثناهای طراحی شده توسط برنامه نویس، کاربرد دارد.
- در مثال اسلاید بعدی، برنامه نویس متدی نوشته که یک آرگومان صحیح را دریافت می کند. اگر مقدار آن کمتر از ۱۸ باشد، استثنای ArithmeticException، پرتاب خواهد شد و در غیراینصورت، پیغامی مبنی بر امکان حق رای چاپ خواهد شد.

مثال throw

```
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...");
  }
}
```

Test it Now

```
Output:Exception in thread main java.lang.ArithmeticException:not valid
```

انتشار استثنا

- استثنا در ابتدا از بالای پشته برنامه ، آخرین متد فراخوانی شده، پرتاب می شود و اگر دریافت (catch) نشود ، در پشته به سراغ متد قبلی می رود 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 (انتهای پشته) برسد. به این روند، انتشار استثنا گفته می شود.
- قاعده: به طور پیش فرض، استثناهای چک نشده در زنجیره فراخوانی انتشار پیدا می کنند.

مثالی از انتشار استثنا

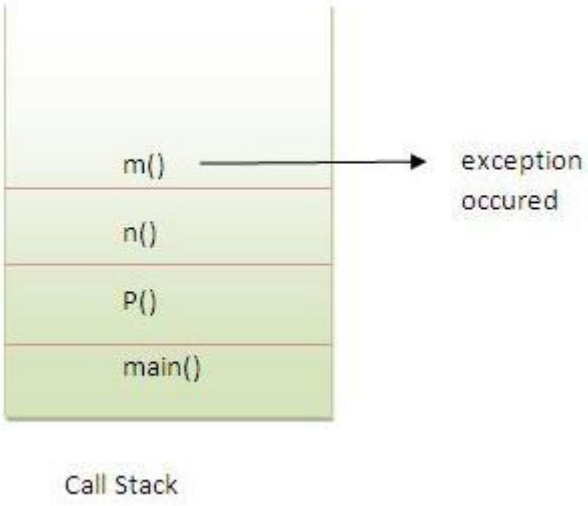
```
class TestExceptionPropagation1{
  void m(){
    int data=50/0;
  }
  void n(){
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   TestExceptionPropagation1 obj=new TestExceptionPropagation1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

Test it Now

```
Output:exception handled
        normal flow...
```

مثالی از انتشار استثنا

- در مثال نشان داده شده در اسلاید قبلی، استثنا در متد m() رخ داده است و در آنجا مدیریت نشده است. در نتیجه در متد فراخواننده (متد قبلی) که n() است، انتشار پیدا می کند که آنجا هم مدیریت نشده است. این انتشار به متد p()می رسد که در آنجا مدیریت صورت گرفته است.
- استثنا می تواند در هر فراخوانی متدی درون پشته فراخوانی، مدیریت شود: مثلا در متد main() یا p() یا n() یا m()



انتشار استثنا

- نکته: به طور پیش فرض، استثناهای چک شده، درون پشته فراخوانی ها انتشار نمی یابند.
- در این مثال به عمد یک استثنای چک شده را به صورت تصنعی ایجاد کرده ایم (با کلمه throw)
- این مثال نشان می دهد استثناهای چک شده، انتشار پیدا نمی کنند.

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

Test it Now

```
Output:Compile Time Error
```

کلمه throws

- کلمه throws برای اعلان یک استثنا استفاده می شود. این کلمه اطلاعاتی به برنامه نویس درباره احتمال بروز یک استثنای چک شده می دهد و در نتیجه برنامه نویس را تشویق می کند برای مدیریت استثنا چاره ای بیندیشد تا جریان عادی برنامه حفظ شود.
- با استفاده از اعلان استثنا (با کلمه throws) به کامپایلر می گوییم بررسی استثنا از زمان کامپایل به زمان اجرا منتقل شود.
- سوال: کدام استثناها باید اعلان شوند؟

پاسخ: استثناهای چک شده، زیرا:

- استثنای چک نشده: تحت کنترل شما است و می توانید کد را تصحیح کنید.
- خطا(error): تحت کنترل شما نیست. برای مثال: VirtualMachineError، StackOverflowError

- **قاعده نحوی throws:**

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

مزیت throws

- با استفاده از throws، استثناهای چک شده نیز در زنجیره فراخوانی ها در پشته برنامه، انتشار می یابند.

مزیت throws–مثال

```
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...");
  }
}
```

Test it Now

```
Output:exception handled
       normal flow...
```

throws

- نکته : اگر متدی را فراخوانی می کنید 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 اعلان کنید. به این ترتیب، استثنای احتمالی در زمان کامپایل بررسی نشده و بررسی آن به زمان اجرا موکول می شود.

حالت اول: استثنا مدیریت شود

- اگر استثنا مدیریت شود، کد چه در صورت وجود استثنا چه درصورت 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...");
  }
}
```

Test it Now

```
Output: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، اگر استثنا رخ دهد، استثنای مربوطه در زمان اجرا ایجاد می شود، زیرا **throws** کاری برای مدیریت استثنا انجام نمی دهد.
- مثال زیر حالتی را نشان می دهد که استثنا رخ نداده باشد.

```
import java.io.*;
class M{
 void method()throws IOException{
  System.out.println("device operation performed");
 }
}


class Testthrows3{
   public static void main(String args[])throws IOException{//declare exception
    Testthrows3 t=new Testthrows3();
    t.method();

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

Test it Now

```
Output:device operation performed
       normal flow...
```

برنامه ای که در آن استثنا رخ داده است

```
import java.io.*;
class M{
 void method()throws IOException{
  throw new IOException("device error");
 }
}


class Testthrows4{
   public static void main(String args[])throws IOException{//declare exception
    Testthrows4 t=new Testthrows4();
    t.method();

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

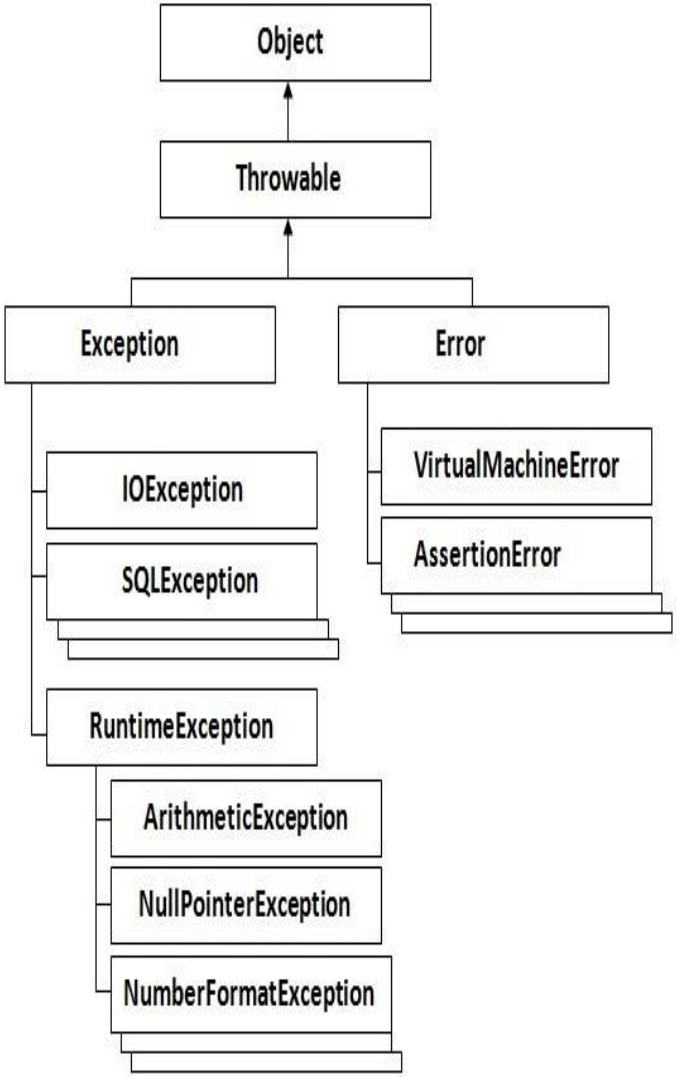
Test it Now

```
Output:Runtime Exception
```

تفاوت میان throw و throws

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 استثنا در هنگام بازنویسی متد



- در هنگام بازنویسی متدها با استفاده از مدیریت استثنائات، از قواعد مختلفی استفاده می شود.
- این قواعد به صورت زیر می باشند:
  - اگر متد کلاس والد، استثنایی را اعلان نکرده باشد، متد بازنویسی شده کلاس فرزند نمی تواند یک استثنای چک شده را اعلان کند، اما می تواند استثنای چک نشده را اعلان نماید.
  - اگر متد کلاس والد، استثنایی را اعلان کرده باشد، متد بازنویسی شده کلاس فرزند می تواند همان استثنای والد را اعلان کند یا استثنایی که در سلسله مراتب استثناها زیرکلاس استثنای کلاس والد است را اعلان کند یا هیچ استثنایی را اعلان نکند. اما نمیتواند استثنایی را اعلان کند که در درخت استثنائات پدر استثنای اعلان شده در کلاس والد بوده است.

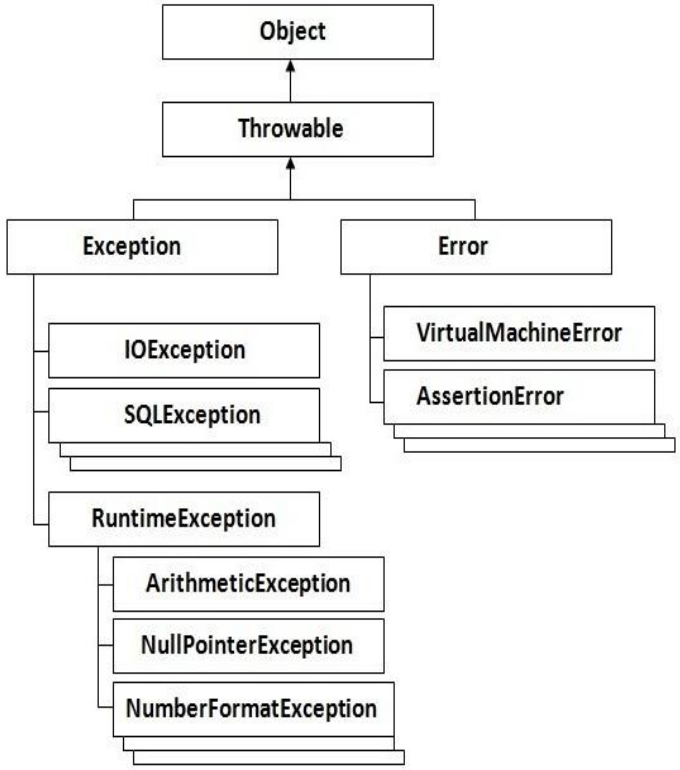
مدیریت استثنا در هنگام بازنویسی متد

- اگر متد کلاس والد، استثنایی را اعلان نکرده باشد:
  - متد بازنویسی شده کلاس فرزند، نمی تواند 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();
  }
}
```

Test it Now

```
Output:Compile Time Error
```



مدیریت استثنا در هنگام بازنویسی متد

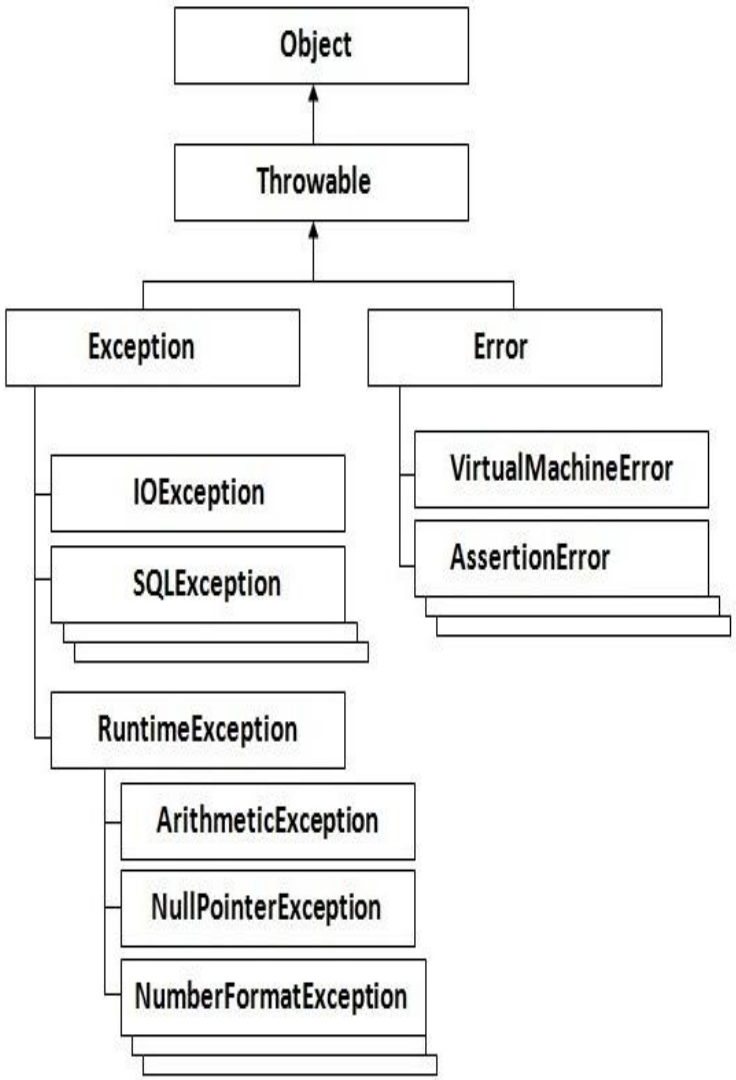
- اگر متد کلاس والد، استثنایی را اعلان نکرده باشد:
  - متد بازنویسی شده کلاس فرزند، نمی تواند یک استثنای چک شده را اعلان کند.
  - اما می تواند یک استثنای چک نشده را اعلان کند.

```
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();
  }
}
```

Test it Now

```
Output:child
```

اگر متد کلاس والد، یک استثنا را اعلان کند



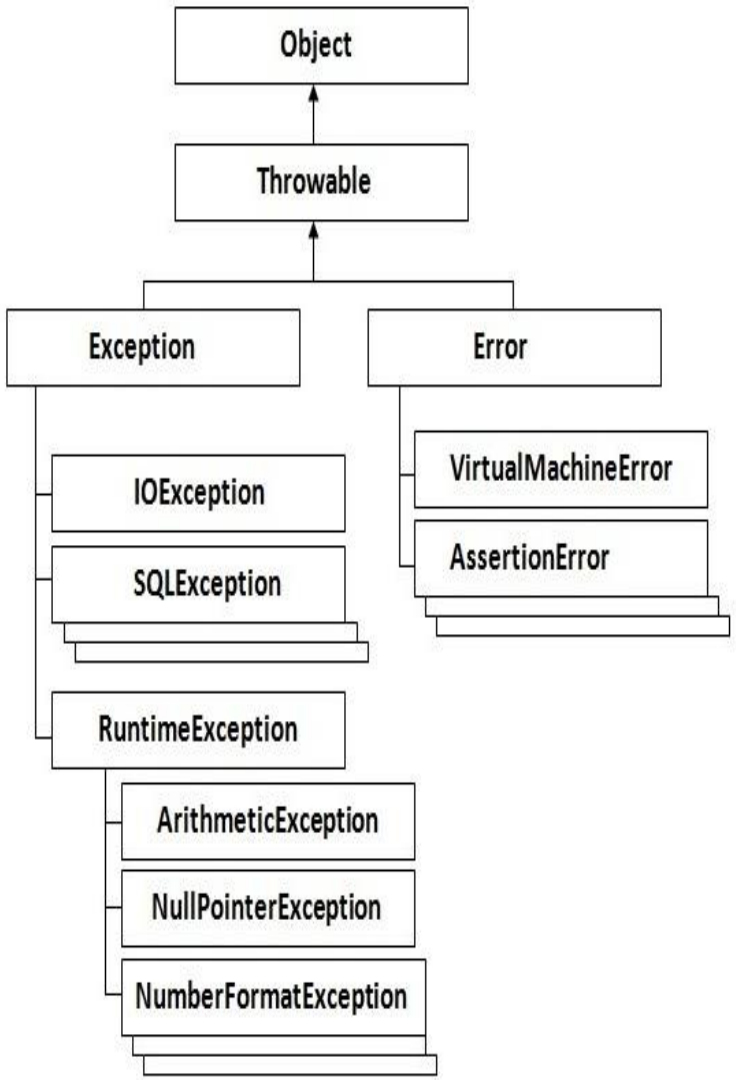
- اگر متد کلاس والد، استثنایی را اعلان کرده باشد، متد بازنویسی شده کلاس فرزند می تواند همان استثنای کلاس والد یا استثنایی را اعلان کند که در درخت استثنائات، فرزند استثنای اعلان شده در کلاس والد باشد یا هیچ استثنایی را اعلان نکند. اما نمی تواند استثنایی را اعلان کند که در درخت استثنائات، پدر استثنای اعلان شده در کلاس والدش باشد.
- در اسلاید بعدی متد بازنویسی شده در کلاس فرزند، استثنایی را اعلان می کند که در درخت استثنائات، پدر استثنای اعلان شده در کلاس والدش است

اگر متد کلاس فرزند، استثنایی را اعلان کند که در درخت استثنائات، پدر استثنای اعلان شده در کلاس والد باشد.

```
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){}
  }
}
```

Test it Now

```
Output:Compile Time Error
```



اگر متد کلاس والد، یک استثنا را اعلان کند

- **متد بازنویسی شده کلاس فرزند، همان استثنای اعلان شده در پدر را اعلان می کند.**

```
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){}
  }
}
```

Test it Now

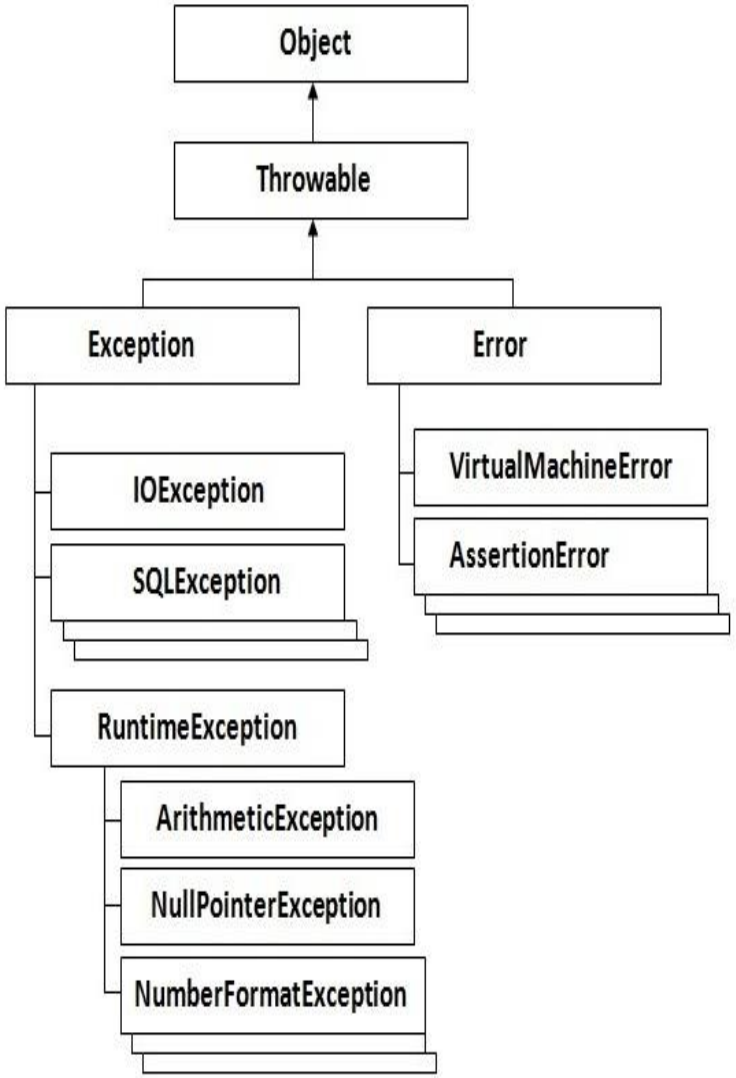
```
Output:child
```

وقتی متد بازنویسی شده در کلاس فرزند، استثنایی را اعلان کند که در درخت استثنائات، فرزند استثنای اعلان شده در کلاس والد باشد.

```
import java.io.*;
class Parent{
  void msg()throws Exception{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){}
  }
}
```

Test it Now

```
Output:child
```



وقتی متد بازنویسی شده در کلاس فرزند، استثنایی را اعلان نکند

```
import java.io.*;
class Parent{
  void msg()throws Exception{System.out.println("parent");}
}

class TestExceptionChild5 extends Parent{
  void msg(){System.out.println("child");}

  public static void main(String args[]){
   Parent p=new TestExceptionChild5();
   try{
   p.msg();
   }catch(Exception e){}
  }
}
```

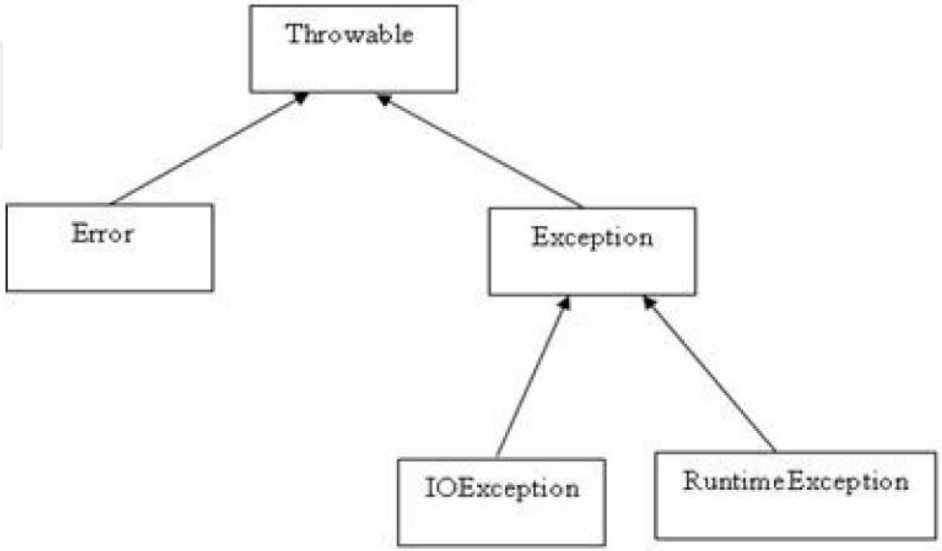
Test it Now

```
Output:child
```

استثناهای طراحی شده توسط برنامه نویس

- استثناهای تعریف شده باید از Throwable ارثبری کنند.
- اگر می خواهید استثنای چک شده ای بنویسید که به طور خودکار با یک ساختار مدیریت استثنا کنترل می شود یا در ابتدای متد اعلان می شود، باید از کلاس exception ارثبری کنید.
- اگر می خواهید استثنای زمان اجرا (چک نشده) بنویسید باید از کلاس RuntimeException ارثبری 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 استثنایی 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 occured: "+m);}

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

Test it Now

```
Output:Exception occured: InvalidAgeException:not valid
       rest of the code...
```

مثالی دیگر از استثنای تعریف شده توسط برنامه نویس

- کلاسهایی که از Exception ارثبری 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.

```
// File Name InsufficientFundsException.java
import java.io.*;

public class InsufficientFundsException extends Exception
{
  private double amount;
  public InsufficientFundsException(double amount)
  {
    this.amount = amount;
  }
  public double getAmount()
  {
    return amount;
  }
}
```

مثالی دیگر از استثنای تعریف شده توسط برنامه نویس

```
// File Name CheckingAccount.java
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;
  }
}
```

- کلاس CheckingAccount دارای متدی به نام withdraw() است که شیئی از کلاس استثنای InsufficientFundsException را پرتاب می کند.

مثالی دیگر از استثنای تعریف شده توسط برنامه نویس

- برنامه زیر متدهای deposit() و withdraw() در checkingAccount را فراخوانی می کند.
- حاصل اجرای این سه فایل:

```
// File Name CheckingAccount.java
import java.io.*;

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

```
// File Name InsufficientFundsException.java
import java.io.*;

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

```
Depositing $500...

Withdrawing $100...

Withdrawing $600...
Sorry, but you are short $200.0
InsufficientFundsException
        at CheckingAccount.withdraw(CheckingAccount.java:25)
        at BankDemo.main(BankDemo.java:13)
```

```
    {
      double needs = amount - balance;
      throw new InsufficientFundsException(needs);
    }
  }
  public double getBalance()
  {
    return balance;
  }
  public int getNumber()
  {
    return number;
  }
}
```

```
    {
      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();
    }
  }
}
```

لیستی از استثناهای چک نشده

Exception	Description
ArithmeticException	Arithmetic 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.
StringIndexOutOfBounds	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.