

کنترل خطا با استفاده از استثنا

فلسفه اصلی جاوا این است که "کدها بد فرم اجرا اجرا نشوند"

معرفی

□ یکی از اهداف اولیه ، ایجاد component های برنامه برای استفاده دیگران است. برای ایجاد یک سیستم قوی و مطمئن، هر یک از component ها نیز باید مطمئن باشند. جاوا با ارائه یک مدل سازگار گزارش خطا با استفاده از استثنا (exception)، به component ها اجازه می دهد تا به صورتی قابل اطمینان کد client را از مشکلات مطلع کند.

□ از اهداف کنترل استثنا (exception handling) آسان کردن ایجاد برنامه های بزرگ و قابل اطمینان با استفاده از کمترین میزان کد است و همچنین اطمینان از این که نرم افزار ایجاد شده دارای خطای پیش بینی و کنترل نشده نیست، افزایش می یابد.

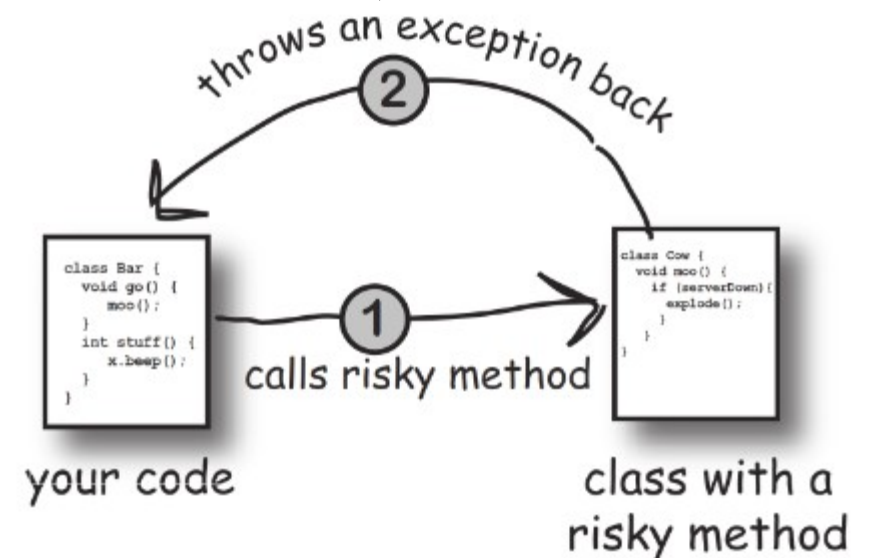
مفاهیم

- با گذشت زمان مشخص شد که برنامه نویسانی که از کتابخانه ها استفاده می کنند تمایل دارند که به صورتی پاسخ و شکست ناپذیر دیده شوند و بگویند: “errorها ممکن است برای بقیه رخ دهند اما نه در مورد کدهای ما”
- مفهوم exception این است که ما به کد مد نظر استثنایی را نسبت می دهیم. هنگامی که مشکلی رخ دهد، ممکن است ندانیم که چگونه آن را برطرف کنیم ولی این آگاهی را داریم که نمی توان به خوبی روال گذشته ادامه داد و فردی در جایی باید مشخص کند که برای ادامه چکار کنیم.
- یکی دیگر از فواید استفاده از استثنا این است که پیچیدگی کد کنترل خطا را کاهش می دهد.

Exception های پایه

❑ شرایط استثنا در واقع مشکلی است که مانع از ادامه متد یا دامنه جاری می شود.

❑ در یک شرایط استثنا، نمی توان پردازش را ادامه داد؛ چرا که در موقعیت فعلی، اطلاعات لازم برای مقابله با مشکل پیش آمده را نداریم.



Exception های پایه

□ هنگامی که استثنایی را پرتاب می کنید چند اتفاق رخ می دهد:

✓ ابتدا object استثنا ایجاد می شود.

✓ سپس مسیر جاری اجرا متوقف می شود و reference ای از object استثنا، از موقعیت فعلی پرتاب می شود.

✓ در این مرحله مکانیزم کنترل استثنا دست به کار می شود و به جستجوی مکانی مناسب برای ادامه دادن اجرای برنامه می پردازد.

Exception های پایه

این عمل پرتاب (throw) کردن یک استثنا نامیده می شود. □

```
if (t == null)
    throw new NullPointerException();
```

آرگومان های exception □

✓ در همه استانداردهای استثنا دو نوع constructor وجود دارد: یک constructor پیش فرض و یک constructor که یک رشته string را به عنوان آرگومان می گیرد و از طریق آن می توان اطلاعات مربوطه را در استثنای مورد نظر قرار داد.

```
throw new NullPointerException("t = null");
```

Exception های پایه

❑ کلیدواژه throw نتایج جالبی را تولید می کند. یک راه ساده برای فهمیدن کنترل

استثنا این است که به آن مانند مکانیزم های مختلف return بنگریم.

❑ به علاوه می توانید هر نوع throwable موجود در کلاس پایه استثنا را پرتاب

نمایید.

❑ اطلاعات مربوط به خطا هم در داخل object استثنا و هم به صورت ضمنی در

نام کلاس استثنای مربوط موجود است.

گرفتن یک استثنا

برای آنکه به چگونگی مهار یک
استثنا پی ببرید (catching)
،ابتدا باید مفهوم منطقه حفاظت
شده (try) را بدانید.



بلاک Try

□ در صورتی که در داخل متدی هستید و exception ای رخ دهد و یا اگر متد دیگری را در طول این متد فراخوانی کرده اید و دچار exception شده است، این متد در طی پردازش استثنا پایان می یابد. در صورتی که نمی خواهید در حین این پردازش از متد مد نظر خارج شوید، می توانید از بلوک try استفاده کنید تا استثنا را مهار کند.

```
try{  
    // Code that might generate exceptions  
}
```

Exception handlers

❑ استثنای پرتاب شده باید بالاخره در جایی پایان یابد. این مکان توسط کنترل گر استثنا مشخص می شود و برای هر نوع استثنا یک catch وجود دارد. کنترل گر استثنا بلافاصله بعد از بلاک try توسط کلید واژه catch مشخص می شود:

```
try {  
    // Code that might generate exceptions  
} catch(Type1 id1){  
    // Handle exceptions of Type1  
} catch(Type2 id2) {  
    // Handle exceptions of Type2  
} catch(Type3 id3) {  
    // Handle exceptions of Type3  
}
```

کنترل گره‌های استثنا

- کنترل گره‌ها باید بلافاصله بعد از بلاک try مشخص شوند.
- در صورتی که استثنایی پرتاب شود توسط اولین بلاک catch ای که آرگومان های آن با نوع استثنای مربوطه منطبق است، گرفته می شود.
- سپس وارد بلاک catch شده و استثنا کنترل می شود. جستجو برای کنترل گره‌ها زمانی که بلاک Catch پایان می یابد، متوقف می شود.
- تنها بلاک catch ای که با استثنا منطبق است اجرا می شود و این دستور مانند دستورات switch نیازی به break در هر case ندارد.

پایان دادن در مقایسه با از سر گیری

□ دو مدل اساسی در نظریه exception-handling وجود دارد:

✓ جاوا "پایان دادن (termination)" به عملیات را پشتیبانی می کند که در آن فرض می شود که خطای ایجاد شده بسیار حیاتی و خطرناک است و هیچ راهی برای بازگشت به نقطه ای که استثنا رخ داده وجود ندارد.

✓ مدل دوم "از سر گیری (resumption)" است و بدین معناست که از کنترل گر خطا انتظار می رود که عملیاتی را برای اصلاح وضعیت انجام دهد و سپس متد معیوب با فرض موفقیت در دومین دفعه، دوباره اجرا می گردد.

استثناهای خود را ایجاد کنید

برای ایجاد کلاس استثنای خودتان باید از کلاس های استثنای موجود ارث ببرید


```
class SimpleException extends Exception {  
}  
public class InheritingExceptions {  
    public void f() throws SimpleException  
    {  
        System.out.println("Throw SimpleException from f()");  
        throw new SimpleException();  
    }  
    public static void main(String[] args) {  
        InheritingExceptions sed = new InheritingExceptions();  
        try {  
            sed.f();  
        } catch(SimpleException e) {  
            System.out.println("Caught it!");  
        }  
    }  
}
```

استثناهای خود را ایجاد کنید

❑ در کنترل گرها یکی از متدهای Throwable (که از آن exception به ارث برده است) متد `printStackTrace()` نامیده می شود.

❑ از طریق خروجی این متود می توان اطلاعاتی را در رابطه با ترتیب متدهایی که فراخوانی شده اند بدست آورد تا به نقطه ای که استثنا رخ داده است برسیم.

استثناها و log گرفتن

همچنین می توان از طریق `java.util.Logging` از خروجی `log` گرفت و اطلاعات مربوطه را ثبت کرد. 

```
class LoggingException extends Exception {  
    private static Logger logger = Logger.getLogger("LoggingException");  
    public LoggingException() {  
        StringWriter trace = new StringWriter();  
        printStackTrace(new PrintWriter(trace));  
        logger.severe(trace.toString());  
    }  
}
```

استثناها و log گرفتن

❑ متد استاتیک `Logger.getLogger()` یک شی `logger` را توسط یک نام ایجاد می کند.

❑ آسان ترین راه برای نوشتن در یک `Logger` این است که متد مرتبط با سطح پیغام `log` گرفتن را فراخوانی کنید؛ متدی مانند `severe()`

تخصیص استثنا

□ زبان برنامه نویسی جاوا شما را تشویق به آگاه کردن برنامه نویسان client (که از متدهای شما استفاده می کنند) در رابطه با استثناهایی که ممکن است در متد شما رخ دهد، می کند.

□ جاوا قواعدی را فراهم می کند (و شما را مجبور به استفاده از این دستورات می کند) که از طریق آن می توانید به برنامه نویس client خود بگویید که متد مد نظر چه استثناهایی دارد تا او نیز بتواند آن ها را کنترل کند. به این کار مشخص کردن استثنا (exception specification) می گویند.

□ تخصیص استثنا از کلید واژه throw استفاده می کند و سپس به دنبال آن لیستی از انواع استثنای ممکن برای متد می آید. بنابراین تعریف متد مد نظرتان می تواند به شکلی مانند دستور زیر باشد:

```
void f() throws TooBig, TooSmall, DivZero
```

تخصیص استثنا

❑ نمی توانید در رابطه با تخصیص استثنا دروغ بگویید. در صورتی که قطعه کدی در متد شما باعث ایجاد استثنا شود و متد مدنظر آن را handle نکند، کامپایلر آن را شناسایی کرده و شما را ملزم به رسیدگی و تخصیص استثنا می کند تا از این طریق استثنا throw شود.

❑ در یک جا می توانید دروغ بگویید: می توانید به شیوه ای ادعا کنید که استثنا را throw کرده اید، در صورتی که در واقع چنین کاری را انجام نداده اید. کامپایلر حرف شما را می پذیرد و تمام استفاده کنندگان از متد شما را ملزم می کند تا به شیوه ای رفتار کنند که انگار آن استثنا throw شده است.

Catch کردن همه استثناها

- ❑ می توان کنترل گری را ایجاد کرد که هم نوع استثنایی را catch کند. این کار با گرفتن نوع استثنای کلاس پایه exception انجام می پذیرد.
- ❑ از این طریق می توان هر exception ای را کنترل کرد. بنابراین اگر بخواهید از آن استفاده کنید، آن را در انتهای لیست کنترل گرها قرار دهید تا از زود اقدام کردن نسبت به کنترل گر استثنایی که ممکن است به شیوه دیگری از آن استفاده می کند، اجتناب کنید.
- ❑ کلاس استثنا از کلاس Throwable به ارث می برد.
- ❑ بعضی متدهای موجود در Throwable عبارتند از:
 - ✓ String getMessage
 - ✓ String getLocalizedMessage: پیغامی در محدوده محلی
 - ✓ String toString: بدست آوردن پیغام جزئیات
 - ✓ void printStackTrace(PrintStream)
 - ✓ void printStackTrace(java.io.PrintWriter)
 - ✓ Throwable fillInStackTrace: () اطلاعات را در شی throwable در رابطه با حالت جاری خانه های stack ذخیره می کند. این متد هنگامی مفید است که نرم افزاری برای بار دوم یک خطا یا استثنا را throw می کند

Rethrow کردن یک استثنا

□ گاهی اوقات شرایطی اتفاق می افتد که استثنایی که قبلاً catch کرده ایم دوباره rethrow می شود. این شرایط به خصوص هنگامی رخ می دهد که از Exception جهت کنترل هر استثنا استفاده کرده ایم.

```
catch(Exception e) {  
    System.out.println("An exception was thrown");  
    throw e;  
}
```

□ همچنین ممکن است که یک استثنای متفاوت را از استثنایی دیگر که catch کرده ایم rethrow کنیم. این عملیات همان تاثیری را دارد که متد fillInStackTrace دارد.

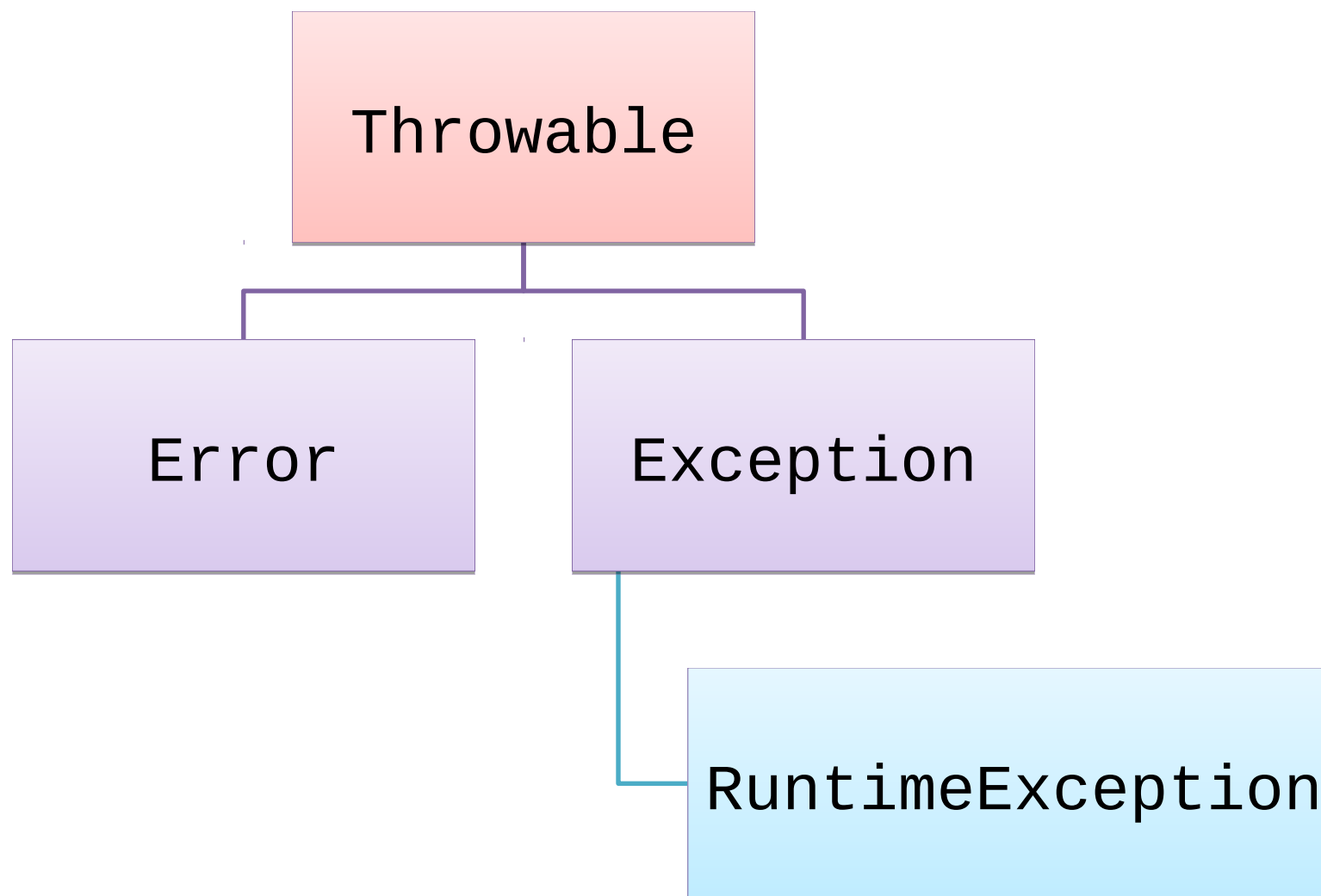
زنجیره های استثنا

❑ در برخی مواقع می خواهیم استثنایی را catch کنیم و استثنای دیگری را throw نماییم اما تمایل داریم اطلاعات مربوط به استثنای ایجاد شده را نیز حفظ کنیم به این کار زنجیره ای شدن استثنا(exception chaining) می گویند.


❑ جالب توجه است که تنها زیرکلاس های throwable که آرگومان های هدف را در constructor فراهم می کنند، سه کلاس استثنای اساسی هستند: Error (توسط JVM جهت گزارش خطاهای سیستم، استفاده می شود)، Exception و RuntimeException


❑ در صورتی که بخواهیم انواع دیگری از استثنا را chain کنیم می توانیم از طریق متد `initCause()` به جای استفاده از constructor این کار را انجام دهیم.


استثنای استاندارد در جاوا



استثنای استاندارد در جاوا

Throwable  هر چیزی را که می تواند به عنوان یک استثنا:
throw شود را توصیف می کند

Error  خطاهای زمان اجرا و سیستم مد نظر را که در رابطه:
با catch کردن آن ها هیچ نگرانی ای نداریم، نشان می دهد

Exception  نوع پایه ای است که می تواند از هر متد:
موجود در کلاس های کتابخانه جاوا، متدهای نوشته شده
توسط برنامه نویس و اتفاقات زمان اجرا throw شود

حالت خاص : RuntimeException

```
if(t == null)  
    throw new NullPointerException();
```

اینکه بخواهید برای هر reference ای که به متد فرستاده می شود، null بودن را چک کنید کمی هولناک به نظر می رسد.

خوشبختانه این کار لازم نیست چرا که این کار بخشی از استاندارد چک کردن در زمان اجراست که توسط زبان جاوا انجام می پذیرد و در صورتی که فراخوانی ای مربوط به reference ای null صورت گیرد، جاوا به صورت خودکار NullPointerException را throw می کند.

هرگز نیازی به نوشتن تخصیص استثنا برای گفتن آنکه یک متد می تواند یک RuntimeException را throw کند (یا هر نوعی که RuntimeException را به ارث می برد) نیست چون این موارد استثنای بدون کنترل هستند.

حالت خاص : RuntimeException

❑ دلیل آن این است که یک RuntimeException نشان دهنده یک خطای برنامه نویسی است و این خطا به صورتی است که:

❑ نمی توانید آن را پیش بینی کنید. به عنوان مثال یه ارجاع null که خارج از کنترل شما است.

❑ شما به عنوان یک برنامه نویس باید داخل کدتان بروز این خطاها را بررسی کنید. یک استثنا که در نقطه ای از اجرای برنامه رخ می دهد معمولا باعث ایجاد مسائل مشکلاتی در نقطه دیگر اجرای برنامه می گردد.

انجام عملیات پاکسازی توسط Finally

□ معمولا قطعه ای از کد وجود دارد که می خواهیم حتما اجرا شود و چه یک استثنا در بلاک try خود throw شود و چه نشود می خواهیم این کد اجرا گردد.

□ برای رسیدن به این هدف از بلاک finally در پایان کنترلر استثنا استفاده می کنیم.

```
try {  
    riskyMethod();  
} catch(A a1) {  
  
} finally{  
    // Activities that happen every time  
}
```

علت استفاده از finally

❑ بلاک finally زمانی استفاده می شود که نیاز داریم چیزی به جز حافظه به حالت اصلی خود برگردد و مانند نوع پاکسازی مثلا در هنگام یک فایل باز یا ارتباط شبکه، یا زمانی که شکلی را در صفحه نمایش کشیده ایم و یا حتی یک تغییر در جهان خارج به صورتی مدل شده، می باشد. در واقع وقتی یک استثنا اتفاق می افتد، پردازش باقی مانده ی کد در متد متوقف می شود و از آن خارج می شود. اگر متد منابع محلی در دست داشته باشد این مسئله ای است، که فقط این متد از آن اطلاع دارد و آن منبع باید از طریق finally آزاد شود.

❑ استفاده از finally در طول return

✓ به دلیل آنکه بلاک finally همیشه اجرا می شود، ممکن است از چندین نقطه در طول یک متد، Return رخ دهد و همچنان تضمین می گردد که عملیات پاکسازی انجام خواهد گرفت.

محدودیت های استثنا

- هنگامی که متدی را override می کنید می توانید استثنایی را throw کنید که توسط ورژن کلاس پایه ی متد تخصیص داده شده است.
- یک constructor می تواند هر چیزی را که می خواهد throw کند، بدون توجه به اینکه constructor کلاس پایه چه چیزی را throw می کند.
- یک constructor کلاس مشتق نمی تواند استثنایی را که توسط constructor کلاس پایه throw شده، catch کند.
- تطبیق استثنا () : هنگامی که استثنایی throw شد، سیستم کنترل استثنا به دنبال "نزدیکترین" کنترل گر با توجه به ترتیب نوشتن آن ها، می گردد.