انجمن جاواکاپ تقدیم می‌کند

دوره برنامه‌نویسی جاوا

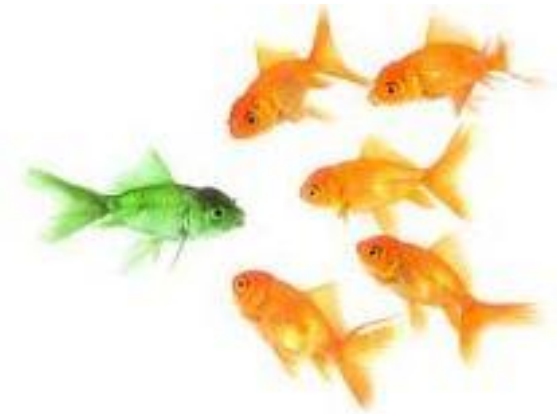
مدیریت خطا و استثنا
Exceptions

صادق علی‌اکبری

حقوق مؤلف

- کلیه حقوق این اثر متعلق به انجمن جاواکاپ است
- بازنشر یا تدریس آن‌چه توسط جاواکاپ و به صورت عمومی منتشر شده است، با ذکر مرجع (جاواکاپ) بلامانع است
- اگر این اثر توسط جاواکاپ به صورت عمومی منتشر نشده است و به صورت اختصاصی در اختیار شما یا شرکت شما قرار گرفته، بازنشر آن مجاز نیست
- تغییر محتوای این اثر بدون اطلاع و تأیید انجمن جاواکاپ مجاز نیست

سرفصل مطالب



- ماهیت خطا و استثنا
- چارچوب مدیریت استثناها
- مزایای چارچوب مدیریت استثناها
- مدیریت استثناها در جاوا

Exception Handling in Java

ماهیت خطا و استثنا

ایراد متد getYear چیست؟

```
public int getYear(String day){
    String yearString = day.substring(0,4);
    int year = Integer.parseInt(yearString);
    return year;
}
```

```
String day = "2010/11/29";
int year = getYear(day);
```

- برای حالت‌های خاص، غیرعادی و غیرمعمول به درستی کار نمی‌کند:
  - پارامتر day ممکن است یک مقدار ناصحیح باشد
    - مثل "salam" که یک تاریخ نیست
  - پارامتر day در قالب موردنظر نباشد
    - مثل "29 Nov 2010"
  - پارامتر day ممکن است یک رشته خالی باشد (" ")
  - پارامتر day ممکن است null باشد

به این حالت‌های خاص، **استثنا (Exception)** گفته می‌شود

مدیریت استثناها

- با یک استثنا چه کنیم؟
  - مثلاً اگر پارامتر یک متد، مطابق انتظار ما نباشد
- اجرای برنامه را قطع کنیم؟
  - تصور کنید یک ورودی اشتباه، باعث قطع کل برنامه شود!
- متد در حال اجرا، خاتمه یابد و مقدار خاصی (مثلا ۱-) را برگرداند
  - شاید مقدار خروجی نداشته باشد (void)
  - یا مقداری به عنوان خروجی «خاص» نتوانیم تعیین کنیم
- خطا را در خروجی نمایش دهیم
  - کدام خروجی؟ شاید برنامه، مبتنی بر وب یا دارای واسط کاربری باشد

هیچ یک از این موارد، راه‌حل مناسبی نیستند

نکته مهم

- گاهی در محل رخداد خطا نمی‌توانیم به خطا رسیدگی کنیم
- حالت غیرعادی را کشف می‌کنیم ولی قادر به پیاده‌سازی عکس‌العمل مناسب نیستیم
- مثال: یک متد کمکی و کتابخانه‌ای را در نظر بگیرید
  - ممکن است در یک برنامه وب یا برنامه کنسول یا برنامه با واسط گرافیکی فراخوانی شود
  - بنابراین این متد نمی‌تواند بازخورد مناسبی هنگام مواجهه با پارامترهای اشتباه ایجاد کند
  - مثال: بازخورد مناسب در برنامه گرافیکی: پنجره خطا، در برنامه کنسول: چاپ خطا و ...
- در این موارد، هنگام برنامه‌نویسی، فقط بروز خطا را گزارش (پرتاب) می‌کنیم
- بخش دیگری از برنامه گزارش خطا را دریافت می‌کند و عکس‌العمل مناسبی اجرا می‌کند

مثال

- متد getYear فقط می‌توانست حالت غیرعادی (استثنا) را تشخیص دهد
- بهتر است این متد بروز استثنا را به متدی که آن را فراخوانی کرده گزارش کند
- و آن متد در این حالت‌های خاص عکس‌العمل مناسبی نشان دهد
- مثلاً پیغام خطای مناسبی به کاربر نشان می‌دهد
- دقت کنید:
  - متد getYear نمی‌داند در چه برنامه و با چه شرایطی فراخوانی شده
  - و در زمان خطا باید چه عکس‌العملی نشان دهد
  - ولی متدی که getYear را فراخوانی کرده، احتمالاً می‌داند

استثنا (Exception) چیست؟

استثنا (Exception) چیست؟

- خطا یا اتفاقی غیرعادی که در جریان اجرای برنامه رخ می‌دهد
- روند اجرای طبیعی برنامه را مختل می‌کند
- مثال:
  - ورودی نامعتبر
  - تقسیم به صفر
  - دسترسی به مقداری از آرایه که خارج محدوده است
  - خرابی هارددیسک
  - باز کردن فایلی که وجود ندارد

رفتار پیش‌فرض جاوا در زمان بروز استثنا

- به صورت پیش‌فرض، اگر در زمان اجرا خطا یا استثنایی رخ دهد:
- این استثنا توسط اجراگر جاوا (JVM) کشف می‌شود
- توضیحاتی درباره این استثنا در خروجی چاپ می‌شود
- اجرای برنامه قطع می‌شود و خاتمه می‌یابد
- اما معمولاً این رفتار پیش‌فرض مناسب نیست
  - برنامه‌نویس باید عکس‌العمل بهتری برای زمان بروز استثنا پیاده‌سازی کند

مثال

```
17    public class DivByZero {
18          public static void main(String a[]) {
19                System.out.println(3/0);
20          }
21    }
```

```
Exception in thread "main"
  java.lang.ArithmeticException: / by zero
at DivByZero.main(DivByZero.java:19)
```

- نکته: استثنا یک مفهوم در زمان اجراست
- این کد هیچ خطایی در زمان کامپایل ندارد

مدیریت استثنا (Exception Handling)

- برای مدیریت و کنترل خطاها و استثناها، چارچوبی وجود دارد
- Exception Handling Framework
- بسیاری از زبان‌های برنامه‌نویسی از این چارچوب کلی پشتیبانی می‌کنند
- C++, Java, C#, ...
- این چارچوب، مدیریت استثناها را ساده می‌کند
- بخش اصلی برنامه را از بخش مدیریت استثناها تفکیک می‌کند
- به این ترتیب: برنامه‌نویسی و فهم برنامه‌ها ساده‌تر می‌شود

چارچوب مدیریت استثنا
Exception Handling Framework

مثالی از مدیریت استثنا در جاوا

```
public class ExceptionHandling {
 public static void main(String[] args) {
   try{
     f();
     g();
   }catch(Exception e){
     System.out.println(e.getMessage());
   }
   m();
 }
 private static void f() {...}
 private static void g() { h(); }
 private static void h() {...}
 private static void m() {...}
}
```

بلوک try-catch

Exception Handler

اگر خطایی در بلوک try-catch (مثلاً در متد f یا g) رخ دهد، روال عادی اجرای برنامه قطع و بخش catch اجرا می‌شود

وقتی یک استثنا رخ می‌دهد، چه می‌شود؟

۱- یک «**شیء استثنا**» ایجاد می‌شود (Exception Object)

۲- شیء استثنا به اجراگر جاوا (JVM) تحویل داده می‌شود

- به این عمل "**پرتاب استثنا**" گفته می‌شود (Throwing an Exception)

- شیء استثنا شامل اطلاعاتی مانند این موارد است:
  - پیغام خطا
  - اطلاعاتی درباره نوع خطا
  - شماره خطی از برنامه که استثنا در آن رخ داده است

وقتی یک استثنا رخ می‌دهد، چه می‌شود؟ (ادامه)

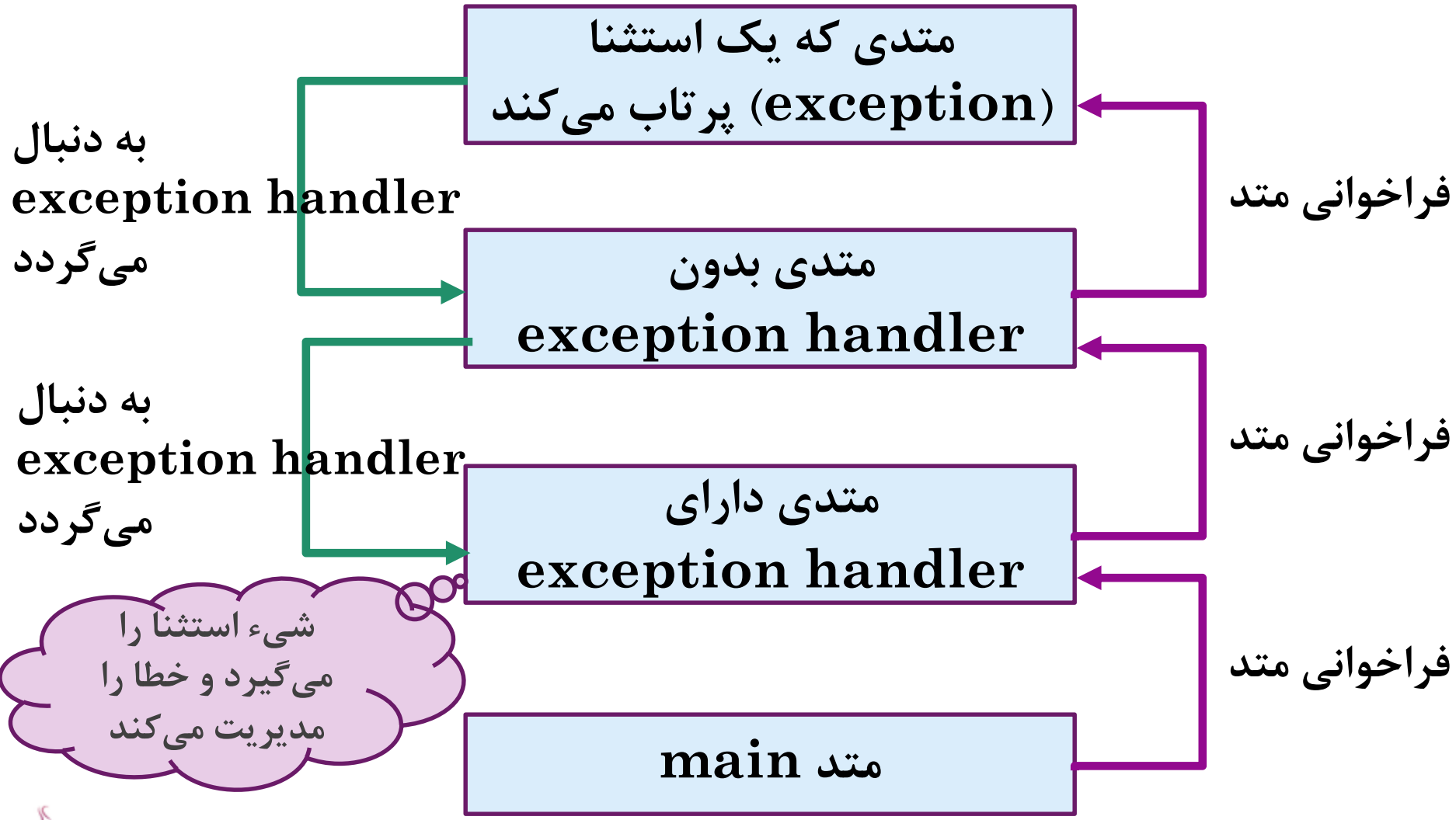
۳- روند اجرای طبیعی برنامه متوقف می‌شود

۴- اجراگر جاوا به دنبال **مسؤول بررسی استثنا** (بخش catch) می‌گردد

- به این مسؤول، **exception handler** می‌گویند
- پشته (stack) فراخوانی متدها را به ترتیب می‌گردد تا این بخش را پیدا کند

- اگر چنین بخشی (exception handler) را پیدا کند:
  - شیء استثنا که پرتاب (throw) شده، توسط این بخش گرفته (catch) می‌شود
  - اجرای برنامه از این بخش ادامه می‌یابد (اجرای طبیعی متوقف شده)
  - از اطلاعات موجود در شیء استثنا برای مدیریت بهتر این حالت خاص استفاده می‌شود
- اگر این بخش نباشد: «رفتار پیش‌فرض جاوا» در مقابله با استثنا اجرا می‌شود
  - (پیغام خطا در خروجی استاندارد چاپ می‌شود و اجرای برنامه خاتمه می‌یابد)

نحوه عملکرد چارچوب مدیریت استثنا



مرور مجدد مثال:

```
public class ExceptionHandling {
 public static void main(String[] args) {
   try{
     f();
     g();
   }catch(Exception e){
     System.out.println(e.getMessage());
   }

  m();
 }
 private static void f() {...}
 private static void g() { h(); }
 private static void h() {...}
 private static void m() {...}
}
```

- اگر در اجرای f خطایی رخ دهد؟
- اگر در اجرای h خطایی رخ دهد؟
- اگر در اجرای m خطایی رخ دهد؟

چارچوب مدیریت استثنا در جاوا

یادآوری: متد getYear

```
public static Integer getYear(String day) {
        String yearString = day.substring(0, 4);
        int year = Integer.parseInt(yearString);
        return year;
}

public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter a well-formed date: ");
      String date = scanner.next();
      Integer year = getYear(date);
      System.out.println(year);
}
```

بازنویسی متد getYear در چارچوب مدیریت استثنا

```
public static int getYear(String day) throws Exception{
      if (day == null || day.length() == 0)
            throw new Exception("Bad Parameter");

      String yearString = day.substring(0, 4);
      int year = Integer.parseInt(yearString);
      return year;
}
```

نحوه استفاده از متد getYear

```
public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);
 System.out.print("Enter a date: ");
 String date = scanner.next();
 try {
    Integer year = getYear(date);
    System.out.println(year);
   } catch (Exception e) {
      System.out.println(e.getMessage());
   }
 }
}
```

کلیدواژه‌های جاوا در چارچوب مدیریت استثنا

- **throw**

```
throw new Exception("Bad Parameter");
```

  - یک استثنا را پرتاب می‌کند

- **throws**

```
int getYear(String d)throws Exception{…}
```

  - اگر متدی احتمال دارد یک استثنا پرتاب کند، **باید** آن را اعلان کند

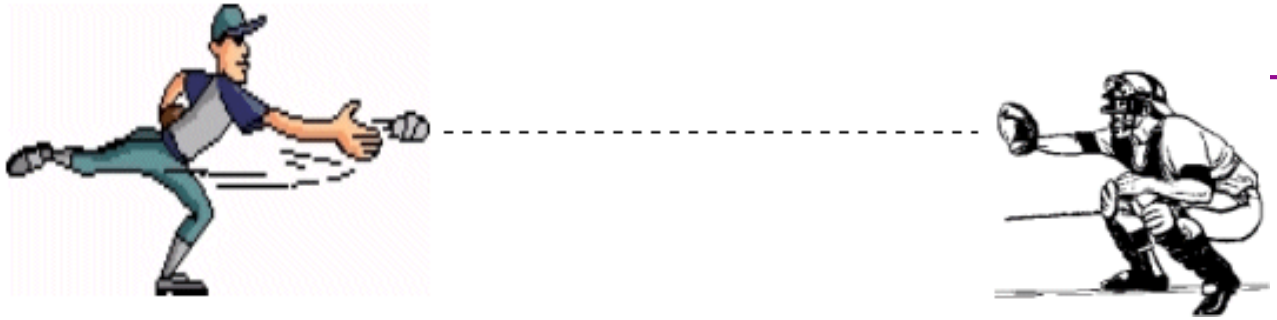
- **try**
  - یک بلوک برای مدیریت استثنا را شروع می‌کند
- **catch**
  - یک استثنا را دریافت و مدیریت می‌کند

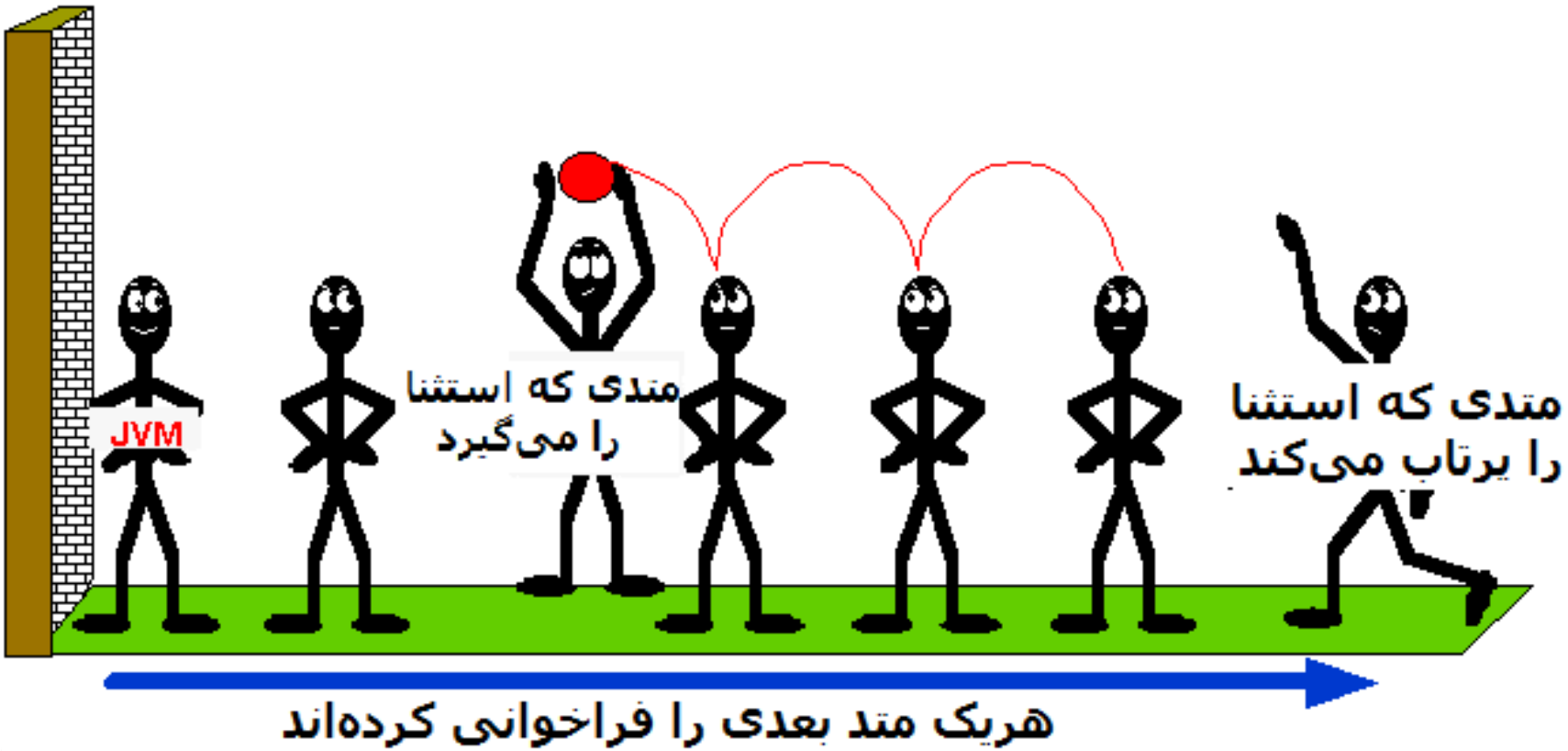
```
try {
   ...
} catch (Exception e) {
   ...
}
```

مثال

```
public class Test1 {
	public static void main(String[] args) {
		try{
			Scanner scanner = new Scanner(System.in);
			int first = scanner.nextInt();
			int second = scanner.nextInt();
			int div = division(first, second);
			System.out.println(div);
		}catch (Exception e) {
			System.out.println(e.getMessage());
		}
	}

	private static int division(int first, int second)
		throws Exception {

		if(second == 0)
			throw new Exception("OOPS! Makhraj Sefre!");
		return first/second;
	}

}
```

کارتون




JVM
متدی که استثنا
را می‌گیرد
متدی که استثنا
را پرتاب می‌کند
هریک متد بعدی را فراخوانی کرده‌اند

مزایای چارچوب مدیریت استثنا

مزایای چارچوب مدیریت استثناها

- جداسازی بخش اصلی برنامه‌ها از کدهای مدیریت خطا و استثنا
- مدیریت خطا در بخشی که این کار امکان‌پذیر است
  - و نه لزوماً در بخشی که خطا رخ داده است
- امکان گروه‌بندی خطاها (استثناها)
  - و مدیریت آن‌ها با توجه به نوع آن‌ها
  - عکس‌العمل مناسب به ازای هر نوع خطا
- نکته:
  - همچنان باید برای **تشخیص**، **گزارش** و **مدیریت** استثناها برنامه‌نویسی کنیم
  - چارچوب مدیریت خطاها مسئول رسیدگی به این امور نیست
  - این چارچوب فقط ما را در سازماندهی مؤثر این کارها کمک می‌کند

جداسازی کدهای مدیریت خطا

- متد شبه کد زیر را در نظر بگیرید
- کل یک فایل را داخل حافظه فراخوانی می‌کند
- (این یک شبه‌کد است، کد جاوا نیست)

```
readFile {
  open the file;
  determine its size;
  allocate that much memory;
  read the file into memory;
  close the file;
}
```

روش سنتی
مدیریت خطاها

```
errorCodeType readFile {
   initialize errorCode = 0;

   open the file;
   if (theFileIsOpen) {
      determine the length of the file;
      if (gotTheFileLength) {
         allocate that much memory;
         if (gotEnoughMemory) {
            read the file into memory;
            if (readFailed) {
               errorCode = -1;
            }
         } else {
            errorCode = -2;
         }
      } else {
         errorCode = -3;
      }
      close the file;
      if (theFileDidntClose && errorCode == 0) {
         errorCode = -4;
      } else {
         errorCode = errorCode and -4;
      }
   } else {
      errorCode = -5;
   }
   return errorCode;
}
```

روش جدید مدیریت خطا

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

این یک شبه‌کد است

(کد جاوا نیست)

که چارچوب مدیریت استثنا را توصیف می‌کند

مفهوم stack trace

- وقتی استثنا را دریافت (catch) می‌کنیم، این اطلاعات در شیء استثنا موجود است:
  - محل اصلی پرتاب شدن استثنا
  - مجموعه (stack) متدهایی که استثنا از آن‌ها رد شده است
- به مجموعه این اطلاعات stack trace گفته می‌شود
- در مواقع اشکال‌یابی برنامه، به این اطلاعات احتیاج داریم
- برخی متدهای دستیابی به stack trace از طریق شیء استثنا:
- printStackTrace();
- getStackTrace();

مثال:

```
public class StackTrace {
  public static void main(String[] args) {
    try{
      f();
    }catch(Exception e){
      e.printStackTrace();
    }
  }
  private static void f() {
    g();
  }
  private static void g() {
    throw new NullPointerException();
  }
}
```

خروجی:

```
java.lang.NullPointerException
at Third.g(Third.java:18)
at Third.f(Third.java:13)
at Third.main(Third.java:5)
```

کوییز

کوییز: خروجی این برنامه؟

```
import java.util.Scanner;
public class Quiz {
 public static void main(String[] args) {
  Scanner scanner = new Scanner(System.in);
  System.out.println("Enter your age:");
  int age = 0;
  boolean validAge = false;
  while (!validAge) {
    try {
      String str = scanner.next();
      age = Integer.parseInt(str);
      validAge = true;
    } catch (Exception e) {
      System.out.println("Enter a valid number:");
    }
  }
  System.out.println("You are "+age+" years old");
 }
}
```

فرض کنید کاربر به ترتیب مقادیر a و abc و ۲۱ را وارد کند

```
Enter your age: a
Enter a valid number: abc
Enter a valid number: 21
You are 21 years old
```

تمرین عملی

تمرین عملی

- استفاده از try ، catch ، throw و throws در یک برنامه
- استفاده از متدهای کلاس Exception
  - e.printStackTrace()
  - e.getMessage();
  - e.getStackTrace()
- صحبت درباره مفهوم لاگ (log)

انواع استثنا
Exception Classes

دسته‌بندی انواع خطاها و استثناها

- هر استثنا، نوعی دارد
  - مثلاً نوع «خطا هنگام خواندن فایل» و «خطای تقسیم بر صفر» متفاوت است
- هر استثنا یک شیء است (شیء استثنا)
- هر شیء نوعی (type یا کلاس) دارد
- بنابراین می‌توانیم استثناها را با کمک نوع آن‌ها دسته‌بندی کنیم
- نوع استثناها به مدیریت بهتر آن‌ها کمک می‌کند
- جاوا کلاس‌های مختلفی برای این منظور دارد
  - مانند NullPointerException یا ClassCastException
- می‌توانیم کلاس‌های جدید استثنا هم ایجاد کنیم
  - مثلاً: IranianBadNationalIdException

مثال

```
private void program()  {
        try{
                int first = readInt();
                int second = readInt();
                int div = division(first, second);
                System.out.println(div);
        }catch (IOException e) {
                System.out.println(e.getMessage());
        }catch (ArithmeticException e) {
                System.out.println(e.getMessage());
        }
}
private int readInt() throws IOException {
        String str = scanner.next();
        if(str.matches("[\\d]+"))
                return Integer.parseInt(str);
        throw new IOException("Bad input");
}
private static int division(int first, int second)
        throws ArithmeticException{

        if(second == 0)
                throw new ArithmeticException("OOPS! Makhraj Sefre!");
        return first/second;
}
```

مثال

```
class MultipleCatch {
  public static void main(String args[]) {
    try {
      int den = Integer.parseInt(args[0]);
      System.out.println(3/den);
    } catch (ArithmeticException e1) {
       System.out.println("Divisor is zero");
    } catch (ArrayIndexOutOfBoundsException e2) {
       System.out.println("Missing argument");
    }
    System.out.println("After exception");
  }
}
```

نحوه ایجاد کلاس Exception جدید

- کلاس جدید باید زیرکلاس **Exception** باشد
  - کلاسی با عنوان java.lang.Exception در جاوا وجود دارد
  - زیرکلاسهای Exception می‌توانند پرتاب (throw) یا دریافت (catch) شوند
- کلاسهای Exception معمولا کلاس‌های ساده‌ای هستند
  - متدها و ویژگی‌های کم و مختصری دارند
  - البته مثل همه کلاس‌ها می‌توانند سازنده، ویژگی و متدهای متنوعی داشته باشند
  - معمولاً یک سازنده بدون پارامتر دارند
  - و یک سازنده با پارامتر رشته دارند که پیغام خطا را مشخص می‌کند

مثال: کلاس java.io.IOException

```
public class IOException extends Exception {

    public IOException() {
        super();
    }
    public IOException(String message) {
        super(message);
    }
    ...
}
```

- مثال از نحوه استفاده از کلاس IOException :

```
if(...)
  throw new IOException();
```

```
if(...)
  throw new IOException("Internal state failure");
```

مثال: ایجاد کلاس استثنای جدید

```
class BadIranianNationalID extends Exception {}
```

```
try {
  if (input.length()!=10) {
    throw new BadIranianNationalID();
  }
  System.out.println("Accept NationalID.");
} catch (BadIranianNationalID e) {
   System.out.println("Bad ID!");
}
```

مرور مجدد متد getYear()

```
public static Integer getYear(String day)
  throws Exception {

  if (day == null)
     throw new NullPointerException();
  if (day.length() == 0)
     throw new EmptyValueException();
  if (!matchesDateFormat(day))
     throw new MalformedValueException();
  String yearString = day.substring(0, 4);
  int year = Integer.parseInt(yearString);
  return year;
}

private static boolean matchesDateFormat(String input) {...}
```

Finally

مفهوم finally

```
try {
    //..
} catch (ExceptionType e) {
   //…
} finally {
   //…
}
```

- بخشی که در finally می‌آید، در انتهای اجرای try-catch حتماً اجرا می‌شود
- اگر خطا پرتاب شود یا نشود، در انتهای کار اجرای بخش finally تضمین می‌شود

بلاک finally

```
try {
    //..
} catch (ExceptionType e) {
   //…
} finally {
   //…
}
```

- این بلاک حتماً اجرا می‌شود
- در هر شرایطی:
  - اتمام طبیعی اجرای بلاک try بدون پرتاب خطا
  - خروج اجباری از بلاک try (مثلاً با return ، break یا continue)
  - خطایی در try پرتاب شود و در catch دریافت شود
  - خطایی در try پرتاب شود و در هیچ یک از بلاک‌های catch، دریافت نشود
  - ...
- بلاک finally برای آزادسازی منابع گرفته شده در try مناسب است
  - مثال: بستن فایل یا اتمام اتصال به دیتابیس
  - البته هر منبعی به جز حافظه. حافظه را زباله‌روب به صورت خودکار آزاد می‌کند

مثال:

```
static void myMethod(int n) throws Exception {
    try {
        switch (n) {
        case 1:
            System.out.println("1st case");
            return;
        case 3:
            System.out.println("3rd case");
            throw new RuntimeException("3!");
        case 4:
            System.out.println("4th case");
            throw new Exception("4!");
        case 2:
            System.out.println("2nd case");
        }
        // continued...
    } catch (RuntimeException e) {
        System.out.print("RuntimeException: ");
        System.out.println(e.getMessage());
    } finally {
        System.out.println("try-block entered.");
    }
}
```

مدیریت استثناها به صورت تودرتو (Nested try-catch)

```
class NestedTryDemo {
  public static void main(String args[]){
    try {
      int a = Integer.parseInt(args[0]);
      try {
        int b = Integer.parseInt(args[1]);
        System.out.println(a/b);
      } catch (ArithmeticException e) {
        System.out.println("Div by zero error!");
      }
    } catch (ArrayIndexOutOfBoundsException e2) {
       System.out.println("Need 2 parameters!");
    }
  }
}
```

کوییز

```
int myMethod(int n) {
 try {
  switch (n) {
  case 1:
    System.out.println("One");
    return 1;
  case 2:
    System.out.println("Two");
    throwMyException();
  case 3:
    System.out.println("Three");
 }
   return 4;
 } catch (Exception e) {
   System.out.println("catch");
   return 5;
 } finally {
   System.out.println("finally");
   return 6;
 }
}
```

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

```
void throwMyException() throws MyException {
      throw new MyException();
}
```

خروجی این قطعه برنامه چیست؟

```
System.out.println(myMethod(1));
System.out.println(myMethod(2));
System.out.println(myMethod(3));
```

پاسخ:

One
finally
6
Two
catch
finally
6
Three
finally
6

تمرین عملی

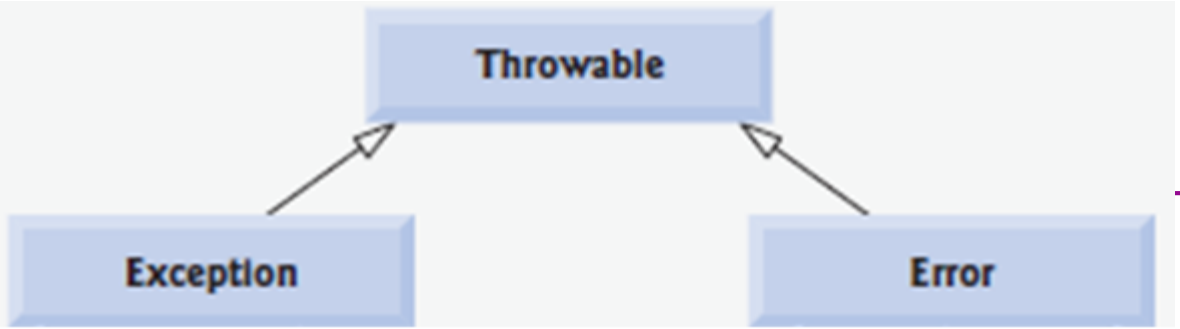
تمرین عملی

- نوشتن یک کلاس استثنای جدید
  - با دو سازنده
- متدی که چند استثنا پرتاب می‌کند
- کنترل کامپایلر و لزوم اعلان استثناهای پرتابی. مثال:
  - throws Exception
  - throws IOException, ArithmeticException
- بلاک finally
  - حتی بدون catch

استثناهای چک‌شده و چک‌نشده

Checked and Unchecked Exception

کلاس‌های استثنا: نگاهی دقیق‌تر

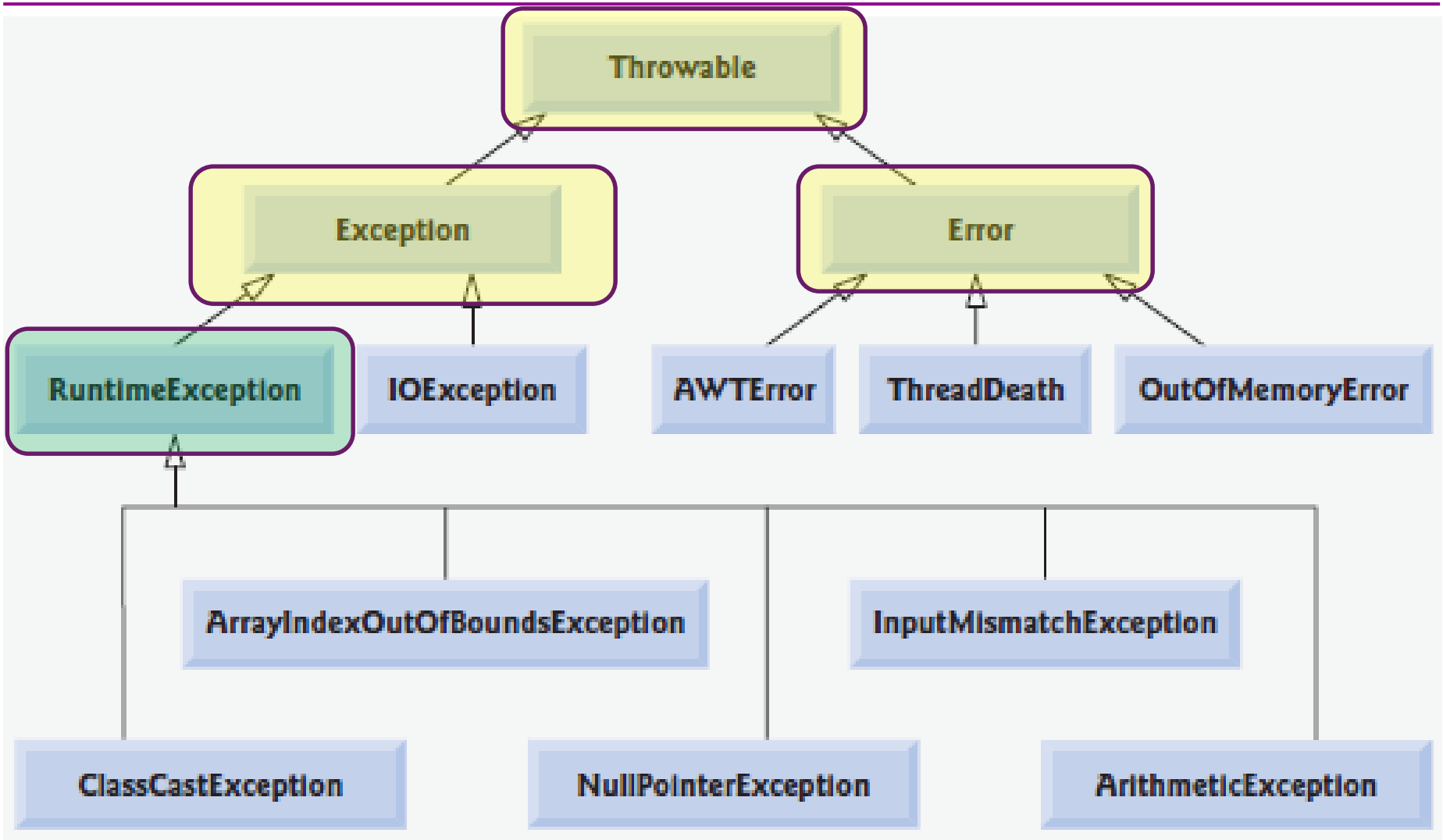


- کلاس Throwable
  - در واقع هر آن‌چه درباره Exception گفتیم، درباره Throwable صادق است
    - مثلاً هر شیء از جنس Throwable قابل پرتاب (throw) یا دریافت (catch) است
- دو نوع Throwable اصلی وجود دارد

۱- Exception : قبلاً دیدیدم (اکثر کلاس‌های استثنا که با آن‌ها سروکار داریم)

۲- Error (خطا) : معمولاً تلاش نمی‌کنیم که آن‌ها را در برنامه catch کنیم
  - حتی اگر آن را catch کنیم، کار مهمی در قبال این خطاها نمی‌توانیم انجام دهیم
  - مانند: OutOfMemoryError

سلسله‌مراتب کلاس‌های استثنا



استثناهای چک‌نشده

```
private static void function(String arg) {
  System.out.println(1 / arg.length());
}
public static void main(String[] args) {
  function("");
}
```

- متد function() ممکن است ArithmeticException پرتاب کند
  - ولی کلیدواژه throws را تصریح نکرده است
- برای بعضی از استثناها، مثل ArithmeticException ، ذکر throws واجب نیست
- اگر این کار اجباری بود، هر متدی که عملگر تقسیم ریاضی داشت باید throws را ذکر می‌کرد
  - با این کار برنامه‌ها پر از throws های نامهم می‌شدند
- انواع استثنای چک‌نشده (Unchecked Exceptions)
- مثل *ArithmeticException* و *ArrayIndexOutOfBoundsException*

استثناهای چک‌شده و چک‌نشده

- **استثناهای چک‌شده (Checked Exception)**
  - **کامپایلر** جاوا بررسی می‌کند:
  - برنامه باید استثنای پیش آمده را دریافت کند یا احتمال پرتاب‌شدن آن را اعلان کند
  - وگرنه، **خطای کامپایلر** رخ می‌دهد
- **استثناهای چک‌نشده (Unchecked Exceptions)**
  - **کامپایلر** دریافت یا اعلان پرتاب را اجبار نمی‌کند (**در زمان کامپایل چک نمی‌شود**)
  - کلاس‌های این نوع، عبارتند از :
  - کلاس Error
  - کلاس RuntimeException
  - زیرکلاس‌های Error و RuntimeException

مثال

```
void example(int x) {
 if(x==1)
   throw new Error();
 if(x==2)
  throw new RuntimeException();
 if(x==3)
  throw new NullPointerException();
 if(x==3)
  throw new IOException();
}
```

بدون اشکال: زیرا RuntimeException و NullPointerException استثناهای چک‌نشده (Unchecked Exception) هستند

```
Syntax Error:
Unhandled Exception Type IOException
```

زیرا IOException یک استثنای چک‌شده (Checked Exception) است

- تصحیح:

```
void example(int x) throws IOException {
   ...
   if(x==3)
     throw new IOException();
}
```

نکته

- مهم: استثناهای چک‌نشده، فقط توسط کامپایلر چک نمی‌شوند
  - رفتار استثناهای چک‌شده و چک‌نشده در زمان اجرا کاملاً مشابه است
  - چک‌نشده، یعنی «چک‌نشده توسط کامپایلر»، در زمان اجرا چک می‌شوند
- اگر می‌خواهید یک نوع Exception جدید ایجاد کنید
  - اگر نمی‌خواهید کامپایلر آن را **چک کند**، آن را چک‌نشده تعریف کنید
  - برای این کار، کلاس جدید را فرزند RuntimeException قرار دهید
- (یادآوری) رفتار کامپایلر درباره استثناهای چک‌شده:
  - اگر متدی ممکن است چنین استثنایی پرتاب کند، باید این مهم را تصریح کند
  - (با کمک دستور throws در ابتدای تعریف متد)

تمرین عملی

تمرین عملی

- نمایش کلاس‌های Error و Exception و Throwable
- استفاده از Throwable به جای Exception
- تعریف کلاس استثنای چک‌نشده
  - مرور رفتار کامپایلر و JVM در قبال استثناهای چک‌شده و چک‌نشده
- مشاهده تعریف برخی استثناهای موجود در جاوا

- RuntimeException
  - NullPointerException, ClassCastException, …
- Error
  - OutOfMemoryError
- Exception

استثناها و موضوع وراثت

کلاسهای استثنا و سلسله مراتب

- در یک عبارت try-catch :
  - اگر در یک بلاک catch یک نوع استثنا را دریافت کنیم،
    نمی‌توانیم در یک catch بعدی زیرکلاس آن نوع استثنا را دریافت کنیم
  - در این صورت، کامپایلر اعلام خطا می‌کند : Unreachable catch block
- مثال:
- چرا؟

```
try {
  int a = Integer.parseInt(args[0]);
  int b = Integer.parseInt(args[1]);
  System.out.println(a/b);
} catch (Exception ex) {
  //..
} catch (ArrayIndexOutOfBoundsException e) {
  //..
}
```

استثناها و ارث‌بری

```
class Parent{
  void f() {...}
}
class Child extends Parent{
  void f() {...}
}
```

- فرض کنید متد f() در زیرکلاس override شده باشد
- f() در زیرکلاس نمی‌تواند استثناهای بیشتری از f() در اَبَرکلاس پرتاب کند
  - انواع استثنا که در متدی در زیرکلاس پرتاب می‌شوند،
    باید کمتر یا مساوی تعریف آن متد در اَبَرکلاس باشد
    (منظور استثناهایی است که با ذکر throws مشخص می‌شوند)
- وگرنه، کامپایلر خطا می‌گیرد

چرا متدی در زیرکلاس نمی‌تواند استثناهای بیشتری پرتاب کند؟

- اگر این قانون وجود نداشت، تعریف کلاس Child بدون خطا می‌شد:

```
class Parent{
  void f() {}
}
class Child extends Parent{
  void f() throws IOException {}
}
```

- در این تعریف، به نوعی رابطه is-a بین Child و Parent نقض شده است

- مثلاً کامپایلر نمی‌تواند متد example را مجبور کند که خطای IOException را catch یا throws کند

```
void example() {
  Parent p = new Child();
  p.f();
}
```

مثال(۱)

```
class Parent{
  void f(){}
}

class Child extends Parent{
  void f()throws Exception{}
}
```

- نتیجه؟

- جواب: خطای کامپایل

مثال(۲)

- نتیجه؟

```
class Parent{
  void f()throws ArithmeticException{}
}

class Child extends Parent{
  void f()
    throws ArithmeticException,
     IOException{}
}
```

- خطای کامپایلر

مثال(۳)

- نتیجه؟

```
class Parent{
  void f()throws ArithmeticException{}
}

class Child extends Parent{
  void f()throws Exception{}
}
```

- خطای کامپایلر

مثال(۴)

- نتیجه؟

```
class Parent{
  void f() throws Exception{}
}
class Child extends Parent{
  void f() throws ArithmeticException{}
}
```

- بدون خطا

امکانات جدید از جاوا ۷

catch چندگانه

```
try{
  f();
}catch (IOException ex) {
  log(ex);
}catch (SQLException ex) {
  log(ex);
}catch (ClassCastException ex){
  throw ex;
}
```

- قبل از جاوا ۷:

- از جاوا ۷ به بعد می‌توانیم:

```
try{
  f();
}catch (IOException | SQLException ex) {
  log(ex);
}catch (ClassCastException ex){
  throw ex;
}
```

امکان try-with-resources

- قبل از جاوا ۷:

```
BufferedReader br = new BufferedReader(new FileReader(path));
try {
    return br.readLine();
} finally {
    if (br != null) br.close();
}
```

- از جاوا ۷ به بعد می‌توانیم:

```
try (BufferedReader br =
           new BufferedReader(new FileReader(path))) {

  return br.readLine();
}
```

- توضیح بیشتر: در مبحث IO

تمرین عملی

تمرین عملی

- رفتار کامپایلر در قبال وراثت و استثناها
- استفاده از امکانات جاوا ۷
  - دریافت ترکیبی
  - try-with-resources : مثال برای Scanner

بِه‌روش‌ها و اشتباههای رایج در کاربرد استثنا

استفاده نادرست از استثناها

- ابزار کنترل جریان اجرای برنامه (Flow Control):
  دستورات شرطی (if) ، حلقه‌ها (مثل for) و ...
- نباید از چارچوب استثناها برای کنترل فرایند اجرا استفاده کنیم
- از Exception فقط برای مدیریت خطا و استثناها استفاده کنید
- مثال از کاربرد نامناسب استثنا:

```
void useExceptionsForFlowControl() {
 try {
   while (true) {
     increaseCount();
   }
 } catch (MaxReachedException ex) {}
// Continue execution
}
```

```
void increaseCount()throws MaxReachedException {
  if (count++ >= 5000)
     throw new MaxReachedException();
}
```

بازپرتاب استثنا و پرتاب استثنای جدید

```
try {
    ...
}catch (IOException ex) {
    ...
    throw ex;
}
```

- گاهی استثنا باید re-throw شود
  - یعنی catch شود، کارهایی انجام شود، و سپس دوباره throw شود

- گاهی هم خطای جدیدی در بلاک catch پرتاب می‌شود
  - یعنی هر کاری که ممکن است در catch انجام می‌دهیم و سپس خطای جدیدی ایجاد و پرتاب می‌کنیم

```
try {
  ...
}catch(IOException e) {
  ...
  throw new ReportDataException(e);
}
```

دریافت (catch) مناسب

```
try {
  db.save(entity);
} catch (SQLException ex) {}
```

- استثنا را نادیده نگیرید
  - مثلاً کد فوق SQLException را خفه می‌کند (کار خوبی نیست)
- به جای دریافت استثناهای کلی (مثل Exception)، استثناهای مشخصی (مثلاً IOException) را دریافت کنید
  - در هنگام اعلان استثناهای پرتابی با کمک throws هم این قاعده را رعایت کنید
- استثنا را در محل مناسب دریافت (catch) کنید
  - اگر در یک محل نمی‌دانید با خطا چه کنید، آن را catch نکنید
  - مثلاً اجازه دهید به متدهای بالادستی (که متد شما را فراخوانده‌اند) پرتاب شود

Throw early catch late

چند نکته

- پیام مناسب و گویا به عنوان message استفاده کنید

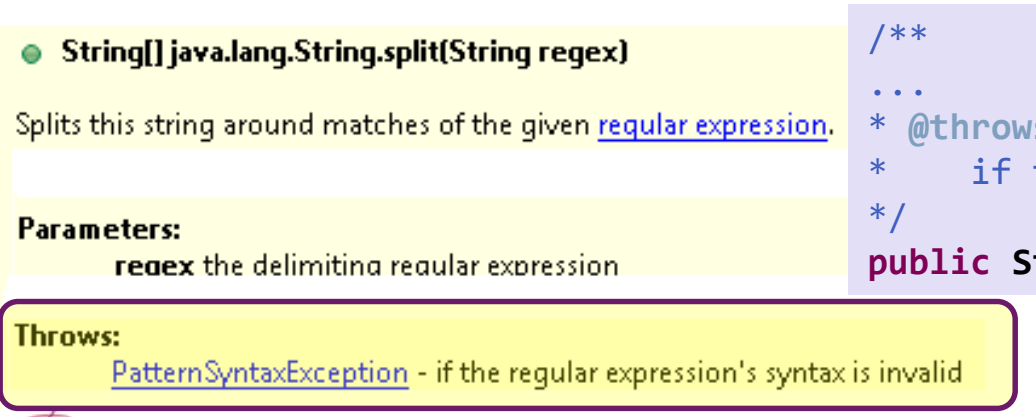
```
throw new IOException(message);
```

- لاگ (Log): دربسیاری از موارد باید بروز خطا را لاگ بزنیم (یعنی این اتفاق را ثبت کنیم)
  - این کار در بلاک catch قابل انجام است

البته بهتر است از فناوری‌های مخصوص لاگ (مثل SLF4J) استفاده کنید

استفاده از System.out.println یا printStackTrace برای این کار مناسب نیست

- مستندسازی مناسب رفتار استثناها در برنامه شما با کمک جاواداک




```
/**
...
* @throws  PatternSyntaxException
*     if the regular expression's syntax is invalid
*/
public String[] split(String regex) {...}
```

جمع‌بندی

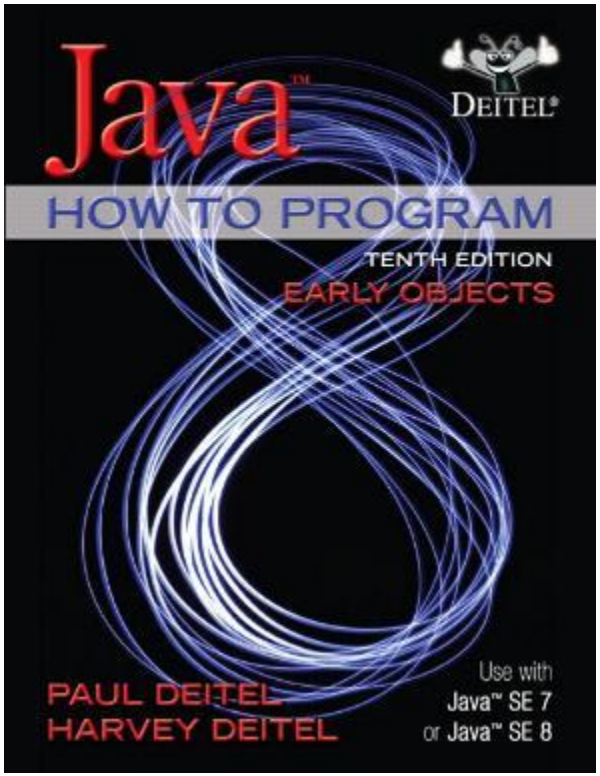
جمع‌بندی

- چارچوب مدیریت خطا در جاوا
  - مزایای این چارچوب
- کلاس Exception
- فرایند ایجاد، پرتاب و دریافت استثناها
- استثناهای چک‌شده و چک‌نشده
- محدودیت‌های تعریف استثنا در زمینه ارث‌بری
  - در تعریف متدهای زیرکلاس

مطالعه کنید

- فصل ۱۱ کتاب دایتل

Java How to Program (Deitel & Deitel)



11 Exception Handling: A Deeper Look

- تمرین‌های همین فصل‌ها از کتاب دایتل

تمرین

- واسط زیر را پیاده‌سازی کنید

```
interface IranValidation {
  void validateNationalID(String ssn)
     throws BadNationalIDException;
  void validatePersianName(String name)
     throws RuntimeException;
  void validateDate(String date)
     throws PersianDateException;
}
```

- از کلاس پیاده‌سازی شده استفاده کنید

جستجو کنید و بخوانید

- موضوعات پیشنهادی برای جستجو:
  - تاریخچه Exception-Handling در زبان‌های دیگر برنامه‌نویسی
  - دستور assert
    - تفاوت آن با JUnit assertions
  - بسیاری از متخصصان معتقدند «استثناهای چک‌شده» تجربه خوبی نبود
    - http://www.mindview.net/Etc/Discussions/CheckedExceptions
  - الگوها و به‌روش‌ها در زمینه استفاده از استثناها
  - مفهوم و کاربرد لاگ (Logging) و فناوری‌های آن (مثلاً SLF4J)

پایان