《Java课程》教案

第13章 正则和异常

[1. 回顾上一章: [分钟] 3](#_Toc499035797)

[2. 课程知识点讲解： 3](#_Toc499035798)

[2.1. 正则的概念和用途 3](#_Toc499035799)

[2.2. 正则元字符 4](#_Toc499035800)

[2.3. String类中的正则用法 5](#_Toc499035801)

[2.4. Pattern和Matcher 6](#_Toc499035802)

[2.5. 异常介绍 8](#_Toc499035803)

[2.6. 异常处理 10](#_Toc499035804)

[3. 本章总结 [分钟] 16](#_Toc499035805)

[4. 测试题 [分钟] 16](#_Toc499035806)

[5. 作业： 16](#_Toc499035807)

[6. 扩展部分： 16](#_Toc499035808)

[7. 学员问题汇总： 17](#_Toc499035809)

编创教师:

课时：分钟

本章主要目标

1.了解正则的概念和用途

2.理解正则元字符

3.String中的正则方法以及Pattern和Matcher使用

4.异常的概述

5.异常的捕获、抛出

6.自定义异常

本章重点

1.String中的正则方法以及Pattern和Matcher的使用

2.异常的捕获、抛出和自定义异常

本章难点

1.正则在java中的使用

2.异常的使用

本章授课注意事项

由实例引出，循序渐进

# 回顾上一章: [分钟]

String、Object、System、Random、Math、Arrays、包装类的使用

例：

String str1 = new String(“abc”);

String str2 = new String(“abc”);

提问：

System.out.println(str1 == str2);

答案：

false

提问：

System.out.println(str1.equals(str2));

答案

true

# 课程知识点讲解：

## 正则的概念和用途

**1）引入**

现实生活中，经常会遇到注册账号的情况，当输入用户名和密码的时候，一般会有类似“6-18位字母数字组合”，或者“6-16位密码，区分大小写”这样的要求，需要我们的账号和密码符合一定的规则，在代码中，我们如何做到检验账户或密码的格式是否符合规范？

j0293236案例讲解

**案例描述：**

以现有的java知识，如何检查账号是否符合“6-18位字母数字组合”的规则

这一规则的含义是：

* 必须是6-18位
* 必须包含数字
* 必须包含字母
* 不能含有除数字和字母之外的其他字符

**案例代码**：（案例/Normal.java）

|  |
| --- |
| public static boolean checkAccount(String account){    if (account.length() < 6 || account.length() > 18) {  return false;  }  boolean hasNumber = false, hasLetter = false;  char[] chars = account.toCharArray();  for (char c : chars) {  if (Character.isDigit(c)) {  hasNumber = true;  }else if (Character.isLetter(c)) {  hasLetter = true;  }else{  return false;//如果包含其他字符，直接返回false  }  }  return hasNumber && hasLetter;    } |

**案例说明**：

（1）规则虽然表述起来比较简单，但是写成代码比较复杂，检验的过程很繁琐，有没有更好的方式来校验这一类的字符串规则？

（2）正则表达式就是我们的解决方案

**2）正则表达式的概念**

* 什么是正则表达式？
  + 简单来讲，正则表达式就是“符合一定规则的字符串”
  + 正则表达式由普通字符和元字符组成，其中元字符的使用是正则表达式的核心
  + 正则表达式常用来检索，替换符合某个模式的文本

j0293236案例讲解

**案例描述：**

上述案例一中的对“6-18位字母数字组合”的校验，使用正则表达式如何完成

**案例代码**：（案例/Normal.java）

|  |
| --- |
| public static boolean checkAccounts(String account){  return account.matches("^(?![0-9]+$)(?![a-zA-Z]+$)[0-9A-Za-z]{6,18}$");    } |

**案例说明**：

（1）上述代码中仅使用了一行代码就完成了校验，其中matchers()方法中传入的参数就是正则表达式

（2）目前先对正则有个感性认识，后面会具体学习写法

**3）正则表达式的用途**

* 正则表达式是一种非常强大的工具，例如：
  + 可以使用正则表达式写网页爬虫，抓取互联网上的数据
  + 可以使用正则表达式写HTML和js的解析器，实现简单的浏览器功能
* 作为程序员，在日常工作中也经常会用到正则表达式：
  + 在程序中校验密码、邮箱、手机号等字符串是否符合规则
  + 在项目或日志文件中搜索和提取你需要的信息
  + Linux命令中大量使用正则表达式对命令结果做筛选和匹配

## 正则元字符

1. **什么是正则元字符，有哪些元字符**

* 正则表达式由普通字符和元字符组成
* 元字符就是正则表达式中具有特殊含义的专用字符
* 元字符非常多，重在理解，不需要去记忆，我们主要讲三种：
  + 用于匹配特定字符的元字符
  + 用于限定字符数量的元字符
  + 用于约束字符边界的元字符
  1. **用于匹配特定字符的元字符：**
* 最常用的是方括号[],例如：
  + [abc]匹配a,b,c中的任意一个；[^abc]匹配任意一个非a,b,c的字符
  + [a-zA-Z]匹配a到z，或A到Z的任意一个字符
* 对于一些常见的字符集，有简单的表示方式：
  + 句号.匹配任意字符
  + \d匹配数字，相当于[0-9]；\D匹配非数字，相当于[^0-9]
  + \w匹配单词字符，相当于[a-zA-Z\_0-9]；\W匹配非单词字符，相当于[^\w]
* 例如：\d[a-z].匹配一个数字 + 一个小写字母 + 任意一个字符
  1. **用于限定字符数量的元字符**
* 最常用的是使用大括号{},如下：
  + {n}恰好出现n次
  + {n,m}至少n次，至多m次；{n,}至少n次，无上限
* 对于常用的数量限定，有简单的表达方式：
  + 问号?表示0次或1次
  + 星号\*表示0次或多次
  + 加号+表示一次或多次
* 例如：a{1,3},可以匹配：a、aa、aaa
  1. **用于约束字符边界的元字符：**
* ^ 表示行的开头
* $ 表示行的结尾
* \b 表示单词的边界
* 例如：对于字符串abc123abc456,使用正则表达式^abc，可以匹配到前面的abc，但是不能匹配到中间的abc
  1. **正则表达式中还能使用逻辑运算|和小括号（）**
* A|B匹配A或者B
* (ab){2}匹配abab

## String类中的正则用法

1. 在java中，使用正则表达式的最简单的方式，是直接用String的方法
2. String类有三种方法支持正则表达式：

* 字符串的匹配：

|  |
| --- |
| public boolean matches(String regex) |

* 字符串的替换：

|  |
| --- |
| public String replaceAll(String regex, String replacement) |

* 字符串的分割：

|  |
| --- |
| public String[] split(String regex) |

C:\Users\Administrator.EZ-20160512UUQE\Documents\Tencent Files\383365267\Image\Group\5RQ3$1@{4_KFUDEWM[~TE66.jpg

将字符串abc12wef333kibd5按照数字分割：

|  |
| --- |
| @Test  public void testStringRegex(){    String str = "abc12wef333kidb5";  String[] result = str.split("\\d+");// “\”用来转义  for (String string : result) {  System.out.println(string);  }    } |
| 输出结果：  abc  wef  kidb |

j0293236案例讲解

**案例描述**：

使用matchers方法，校验字符串是否符合“账号由6-18位字母数字组合”

**案例代码**：（源文件名称）

|  |
| --- |
| 使用多个正则表达式完成这一工作：  public static boolean checkAccount1(String account){    return account.matches("[a-z0-9A-Z]{6,18}") && //6-18位数字或字母  !account.matches("[a-zA-Z]+") && //不能仅仅是  !account.matches("[0-9]+"); //不能仅仅是数字    } |
| 使用一个正则表达式完成这一工作：  public static boolean checkAccounts(String account){  return account.matches("^(?![0-9]+$)(?![a-zA-Z]+$)[0-9A-Za-z]{6,18}$");    } |

**案例说明**：

1）使用一个正则表达式完成校验的方式中，第一个括号里的内容表示后面不能全是数字，第二个括号里的内容表示后面不能全是字母，其中(?!pattern) 这种形式的正则表达式，代表字符串中的一个位置，**紧接该位置之后**的字符序列**不能匹配pattern**

## Pattern和Matcher

1. **Java正则表达式的核心API在java.util.regex包中，有两个关键类：**

* Pattern 用来编译一个正则表达式
* Matcher用来执行正则表达式的匹配操作

1. **String类提供的正则相关方法，内部都是使用Pattern和Matcher实现的：**

|  |
| --- |
|  |

1. **Pattern和Matcher的标准用法有三步：**

* 使用Pattern来编译正则表达式：

|  |
| --- |
| Pattern pattern = Pattern.compile(“\\d+”); |

* 针对目标文本创建一个Matcher

|  |
| --- |
| Matcher matcher = pattern.matcher(“abc23we99nbkd3”); |

* 执行想做的操作，例如：

|  |
| --- |
| 替换：String result = matcher.replaceAll("#");//将数字都替换成#号 |
| 输出结果：abc#we#nbkd# |

1. **Matcher类包含的一些常用方法：**

* Matcher.matches()方法用来匹配整个字符串，作用和String.matches()相同

|  |
| --- |
| @Test  public void testMatchesMethod(){  Pattern pattern = Pattern.compile("(abc){2}");  Matcher matcher = pattern.matcher("abcabc");  boolean matches = matcher.matches();  System.out.println(matches);//输出true  } |

* lookingAt()方法从头开始检测字符串中是否有匹配的子串，找到返回true
* find()方法搜索下一个匹配的子串，找到返回true
* 使用find()匹配成功后，可以用start()和end()拿到子串的位置

|  |
| --- |
| @Test  public void testFind(){    Pattern pattern = Pattern.compile("[a-z]+");  String str = "The game is over.";  Matcher m = pattern.matcher(str);  while(m.find()){  //等同于m.group();  String s = str.substring(m.start(),m.end());  System.out.println(s + "-");    }  } |
| 输出：  he-  game-  is-  over- |

C:\Users\Administrator.EZ-20160512UUQE\Documents\Tencent Files\383365267\Image\Group\5RQ3$1@{4_KFUDEWM[~TE66.jpg

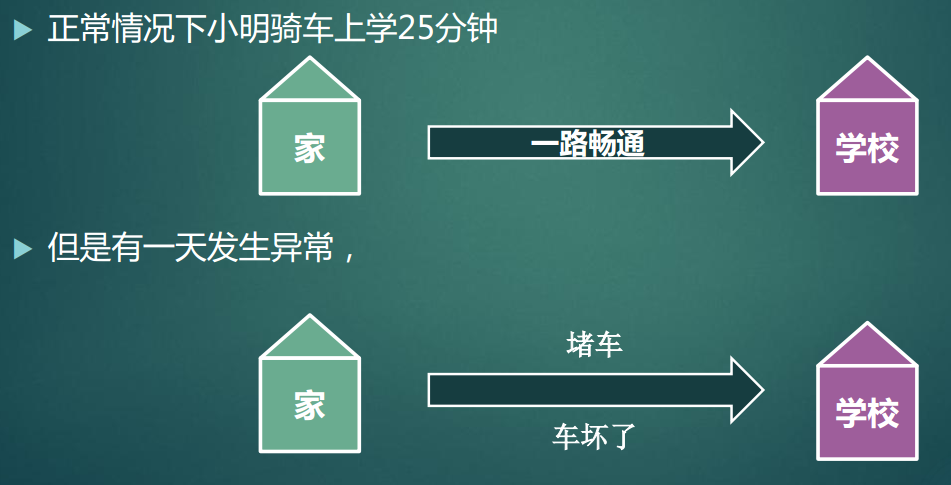
使用Pattern和Matcher，判断字符串“012345”是否符合“6位数字，首位不能为0”的规范

|  |
| --- |
| //练习题：6位数，首位不能为0  @Test  public void testAccount(){    Pattern pattern = Pattern.compile("^[1-9]\\d {5}$");    String str = "012345";  Matcher matcher = pattern.matcher(str);  boolean matches = matcher.matches();  System.out.println(matches);  } |

## 异常介绍

1. **引入**

日常生活中，经常遇到一些情况，使得事情不能按照我们预期的计划进行，比如：



同理，在程序中，也会遇到一些情况，使得我们的程序异常终止：

示例1：

|  |
| --- |
| public static void main(String[] args) {    String[] strs = {"a","b","c"};  System.out.println(strs[3]);    } |
| Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3  at exceptions.TestException.main(TestException.java:10) |

示例2：

|  |
| --- |
| public static void main(String[] args) {    String str = "abc";  int i = Integer.parseInt(str);  System.out.println(i);    } |
| Exception in thread "main" java.lang.NumberFormatException: For input string: "abc"  at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)  at java.lang.Integer.parseInt(Integer.java:580)  at java.lang.Integer.parseInt(Integer.java:615)  at exceptions.TestException.main(TestException.java:9) |

1. **在程序中出现这种“异常”该怎么处理？**

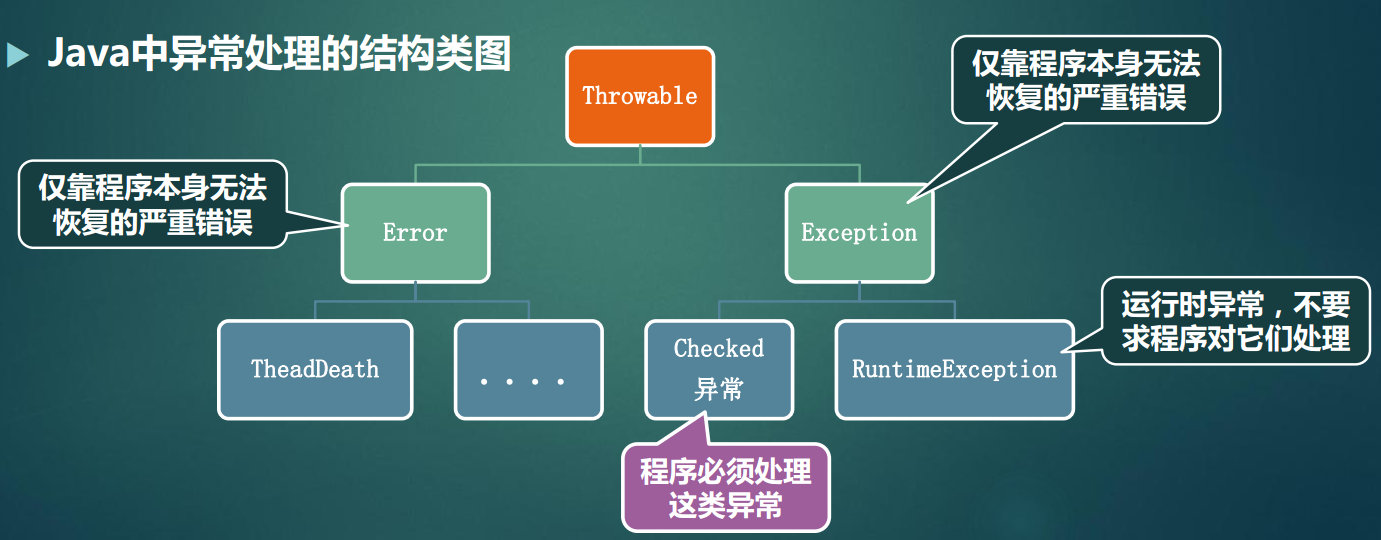
* 是否可以通过if-else解决？

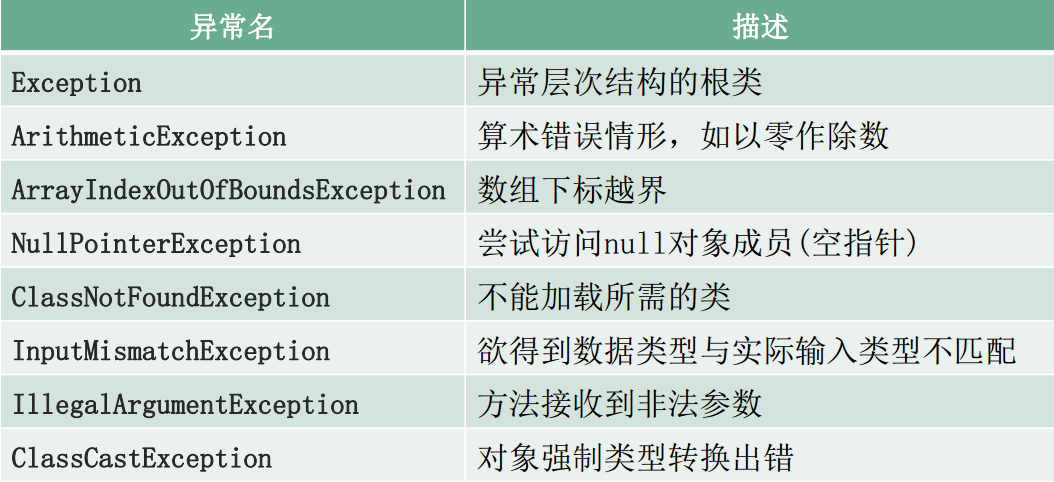
|  |
| --- |
| 针对示例1：  public static void main(String[] args) {    String[] strs = {"a","b","c"};  int i = 3;  if (i >= 3) {  System.out.println("超出数组下标！");  }else{  System.out.println(strs[i]);  }    } |
| 针对示例2：  public static void main(String[] args) {    String str = "abc";    Pattern pattern = Pattern.compile("^[0-9]\*$");  if(pattern.matcher(str).matches()){  System.out.println(Integer.parseInt(str));  }else{  System.out.println("不可转为数字！");    }  } |

* 可见，出现这种异常，如果仅仅通过if-else来解决是不可行的
  + 程序员需要花很大精力去堵漏洞，而且很难堵住所有漏洞
  + 代码臃肿，不利于后期维护
* 针对上述问题，Java提供了解决方案——**异常机制**

1. **认识异常？**

* Java语言使用异常处理机制为程序提供了处理错误的能力
* Java中异常处理的结构类图



* Exception类表示程序可以处理的异常，这类异常可以分为如下两类：
  + 运行时异常
    - Java在运行时才能发现、捕获进行处理的异常
    - RuntimeException及其所有子类均属于运行时异常
  + Checked异常
    - Java在编译时就可以发现的异常，需要强制处理才能运行程序
    - Exception除了RuntimeException之外的所有子类均属于已检查异常
* 常见的异常类型：
  + 

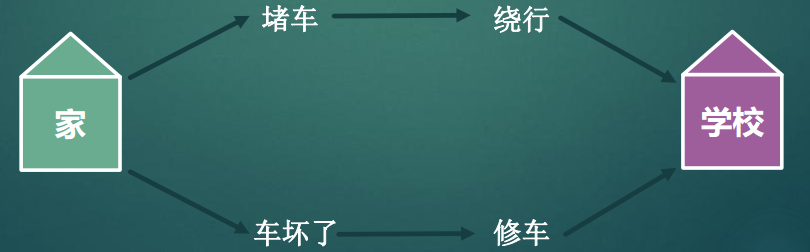
## 异常处理

1. **异常的捕获**

* 程序中出现异常会中断正在运行的程序



* 面对异常该怎么办？通常情况下：



* 那么，Java中是如何处理异常的呢？
  + Java语言使用异常处理机制为程序提供了处理错误的能力：
* Java中使用try、 catch、 finally、 throws、 throw关键字来处理异常
* Java程序处理异常时有两种处理方式：
  + 捕获异常：
    - 尝试捕获，本地处理
    - 使用try、catch、finally关键字
  + 抛出异常
    - 暂不处理，向上抛出
    - 使用throws和throw关键字
* 使用try-catch捕获异常
  + 使用try-catch捕获异常时，异常类型应该更精确，不应该直接写Exception类
  + 语法格式如下：

|  |
| --- |
| **try {** // 可能产生异常的代码块  } **catch (**异常类型 **e) {** //对异常的处理代码  } |

* + 实例：

|  |
| --- |
| **int[] arr = new int[4]; try {**  System.*out.println(arr[4]);* } **catch (ArrayIndexOutOfBoundsException e) {**  System.*out.println("*数组下标越界了*");* } |
| 输出结果：数组下标越界了 |

* Catch块中的异常处理：
  + 用户输出自定义异常处理信息：

|  |
| --- |
| System.err.println(); |

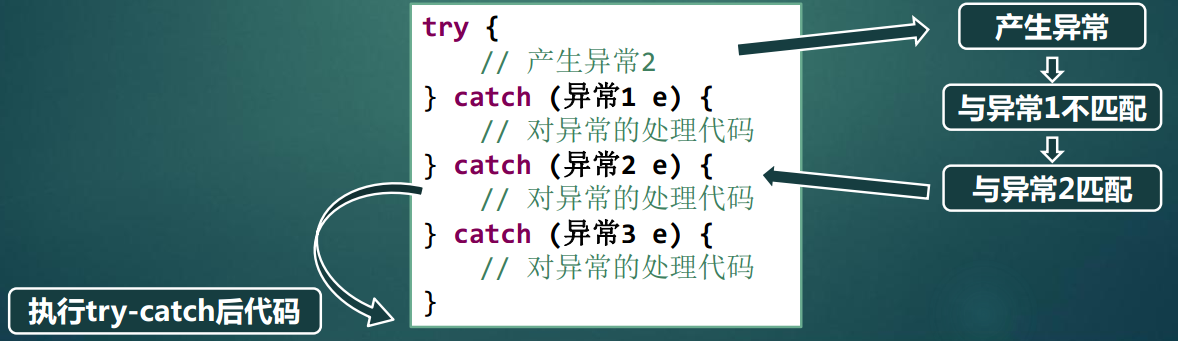
* + 调用异常对象的方法输出异常信息：

|  |
| --- |
|  |

* + 示例：

|  |
| --- |
|  |
| **控制台输出内容：**  数组下标越界  4  java.lang.ArrayIndexOutOfBoundsException: 4  at exceptions.TestHandleException.test1(TestHandleException.java:13)  ……  …… |

* 多重catch块：
  + 一段代码可能会引发多种类型的异常，此时可以使用多重catch捕获异常
    - 当异常发生时，会按顺序查看每个catch语句，并执行第一个与异常类型匹配的catch语句
    - 执行其中一条catch语句后，其后的catch语句将被忽略



* + 示例

|  |
| --- |
| **while (true) {** Scanner in = **new Scanner(System.*in);*** System.*out.print("*请输入除数*:");*  **try { int num = in.nextInt(); int a = 300 / num;** System.*out.println("*商是*:" + a);*  } **catch (InputMismatchException e) {** System.*err.println("*除数必须是整数*");*  } **catch (ArithmeticException e) {** System.*err.println("*除数不能为零*");*  } **catch (Exception e) {** System.*err.println("*其它错误*");*  }  } |
| 输出结果  请输入除数:a  除数必须是整数  请输入除数:3  商是:100  请输入除数:0  除数不能为零  请输入除数: |

* + 思考上面的例子，catch块中的异常类型可否先后调换？
* try-catch-finally
  + 在try-catch后添加finally块，可以确保无论是否发生异常，finally块中的代码**总能被执行**

|  |
| --- |
| 语法格式： **try {**  // 可能产生异常的代码块 } **catch (Exception e) {**  // 对异常的处理代码 } **finally {**  // 有无异常都执行代码块 } |
| 示例：  **int[] arr = new int[4]; try {**  System.*out.println(arr[4]);* } **catch (ArrayIndexOutOfBoundsException e) {**  System.*err.println("*数组下标越界*");*  System.*err.println(e.getMessage());* } **finally {**  System.*out.println("*感谢使用本程序*");* } |
| 输出结果：  数组下标越界 4  感谢使用本程序 |

C:\Users\Administrator.EZ-20160512UUQE\Documents\Tencent Files\383365267\Image\Group\5RQ3$1@{4_KFUDEWM[~TE66.jpg构建一个类型转换异常的例子，并使用try-catch-finally进行处理（使用多重catch块）

1. **异常的抛出**

* 通过throws声明某个方法可能抛出的各种异常
  + 可以同时声明多个异常，之间由逗号隔开
* 语法格式：

|  |
| --- |
| **方法名() throws 异常类型1，异常类型2** |
| 示例1：  **public static void divisor() throws Exception {** Scanner in = **new Scanner(System.*in);*** System.*out.print("*请输入除数*:");* **int num = in.nextInt(); int a = 300 / num;** System.*out.println("*商是*:" + a);*  }  **在方法名divisor()后面，抛出异常，让调用方处理** |
| 示例2：如果一个方法调用了上面的divisor方法，可以捕获这个异常，在本地处理，也可以对异常不捕获，继续抛出，让后面的调用者处理：   * 捕获：   **public static void main(String[] args) { try {**  *divisor();* } **catch (Exception e) {**  e.printStackTrace(); }  }   * 继续抛出：   public static void main(String[] args) throws Exception {  divisor();  } |

* 除了系统自动抛出异常外，有些问题需要程序员自行抛出异常

|  |
| --- |
| **public static void sum() throws Exception {** Scanner in = **new Scanner(System.*in);*** System.*out.print("*请输入一个数*:");* **if (in.hasNextInt()) {**  System.*out.println(in.nextInt());* } **else {  throw new Exception("**输入的必须是一个数字**");**  // 不能有其它代码 }  } |

* + 需要注意的是，在同一个{ }内的抛出异常语句后不能其它有代码
* throw和throws的区别：

|  |
| --- |
|  |

* 最后，种种不同的异常处理方式，该如何选择？
  + 如果功能内部可以解决问题，则try解决
  + 如果处理不了的，throws交给调用者处理
  + 例如：
    - 感冒了——健康发生了异常
    - 自己吃点药——自己try-catch解决
    - 吃了药不管用——throws抛出给医院解决

1. **自定义异常**

* Java中的异常类在某些情况下并不符合我们需求，这时需要我们自定义异常类
* 如何自定义异常类？

|  |
| --- |
| **class PasswordErrException extends Exception { public PasswordErrException(String str) { super(str);** }  } |

* 如何使用自定义异常？

|  |
| --- |
| **try { throw new PasswordErrException("**密码错误**");** } **catch (PasswordErrException e) {** e.printStackTrace();  } |

# 本章总结 [分钟]

正则：

理解正则表达式的概念和作用

掌握基本的正则元字符

会使用Java正则API做字符串的匹配, 替换和分割

异常：

掌握Java中异常的分类

掌握异常的捕获的几种方式

掌握异常的抛出

掌握自定义异常的定义和使用

# 测试题 [分钟]

* 测试题1：
  + 写出账号：字母开头，5-15个字符，允许字母，数字和下划线的正则表达式
  + ^[a-zA-Z][a-z\d\_]{4,14}$
* 测试题2：
  + 写出二代身份证：共18位，前17位位数字，最后一位是数字或字母X
  + ^(\d{6})(\d{4})(\d{2})(\d{2})(\d{3})([0-9]|X)$
* 测试题3：
  + 对常见的checked异常做抛出和捕获两种方式的处理

# 作业：

习题1：



学生信息登记系统中要登记录入身高、体重、年龄，有以下限制，如录入不符合要求就报异常

* 身高：140cm-220cm
* 体重：40kg-100kg
* 年龄：20-35岁

# 扩展部分：

* 扩展部分1：
* 扩展部分2：

# 学员问题汇总：

* 学员问题1：
* 学员问题2：