**Java入门第三季**

**第一章 异常与异常处理**

[**1-1 Java异常简介**](#one_one)

[**1-2 Java中使用try…catch...finally实现异常处理**](#one_two)

[**1-3 Java中通过案例学习try…catch…finally**](#one_two)

[**1-4 练习题**](#one_four)

[**1-5 Java中的异常抛出以及自定义异常**](#one_four)

[**1-6 练习题**](#one_four)

[**1-7 Java中的异常链**](#one_seven)

[**1-8 练习题**](#one_seven)

[**1-9 经验总结**](#one_seven)

**第二章 认识Java中的字符串**

[**2-1什么时候Java中的字符串**](#two_one)

[**2-2 Java中字符串的不变性**](#two_one)

[**2-3 Java中的String类的常用方法Ⅰ**](#two_three)

[**2-4 Java中的String类的常用方法Ⅱ**](#two_three)

[**2-5 认识Java中的StringBuilder类**](#two_three)

[**2-6 Java中的StringBuilder类的常用方法**](#two_six)

[**2-7 练习题**](#two_seven)

**第三章 Java中必须了解的常用类**

[**3-1 Java中的包装类**](#two_seven)

[**3-2 Java中基本类型和包装类之间的转换**](#three_two)

[**3-3 练习题**](#three_three)

[**3-4 Java中基本类型和字符串之间的转换**](#three_three)

[**3-5 练习题**](#three_three)

[**3-6 使用Date和SimpleDateFormat类表示时间**](#three_six)

[**3-7 Calendar类的应用**](#three_seven)

[**3-8 使用Math类操作数据**](#three_eight)

**第四章 Java中的集合框架（上）**

[**4-1 Java中的集合框架概述**](#four_one)

[**4-2 Collection接口 & List接口简介**](#four_two)

[**4-3 学生选课—创建学生类和课程类**](#four_three)

[**4-4 学生选课---添加课程Ⅰ**](#four_three)

[**4-5 学生选课---添加课程Ⅱ**](#four_three)

[**4-6 学生选课—课程查询**](#four_six)

[**4-7 学生选课—课程修改**](#four_seven)

[**4-8 学生选课—课程删除**](#four_seven)

[**4-9 学生选课---应用泛型管理课程Ⅰ**](#four_nine)

[**4-10 学生选课---应用泛型管理课程Ⅱ**](#four_nine)

[**4-11 学生选课---通过Set集合管理课程**](#four_eleven)

**第五章 Java中的集合框架（中）**

**5-1 Map & HashMap 简介**

**5-2 学生选课---使用Map添加学生**

**5-3 学生选课---删除Map中的学生**

**5-4 学生选课---修改Map中的学生**

**第六章 Java中的集合框架（下）**

**6-1 学生选课---判断List中课程是否存在**

**6-2 学生选课---判断Set中课程是否存在**

**6-3 学生选课---获取List中课程的位置**

**6-4 学生选课---判断Map中是否包含指定的key和value**

**6-5 应用Collections.sort() 实现List排序**

**6-6 学生选课---尝试对学生序列排序**

**6-7 Comparable & Comparator**

**6-8 学生选课---实现学生序列排序**

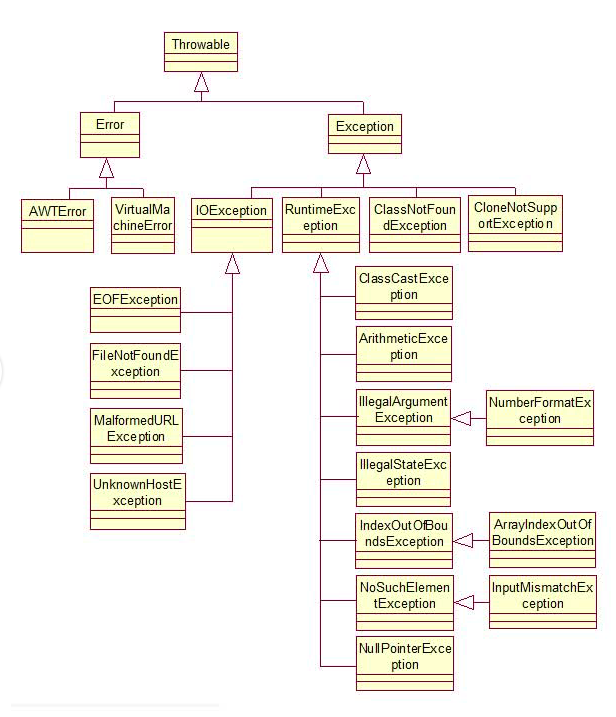
**第七章 综合实战**

**7-1 建议扑克牌游戏**

**第一章 异常与异常处理**

**1-1 Java异常简介**

一般Error出现 , 程序就彻底崩溃了



**1-2 Java中使用try…catch...finally实现异常处理**

处理异常

try-catch以及try-catch-finally

try{

//一些会抛出异常的方法

}catch(Exception e){

//处理该异常的代码块

}catch (Exception2 e){

//处理Exception2的代码块

}…（n个catch块）…{

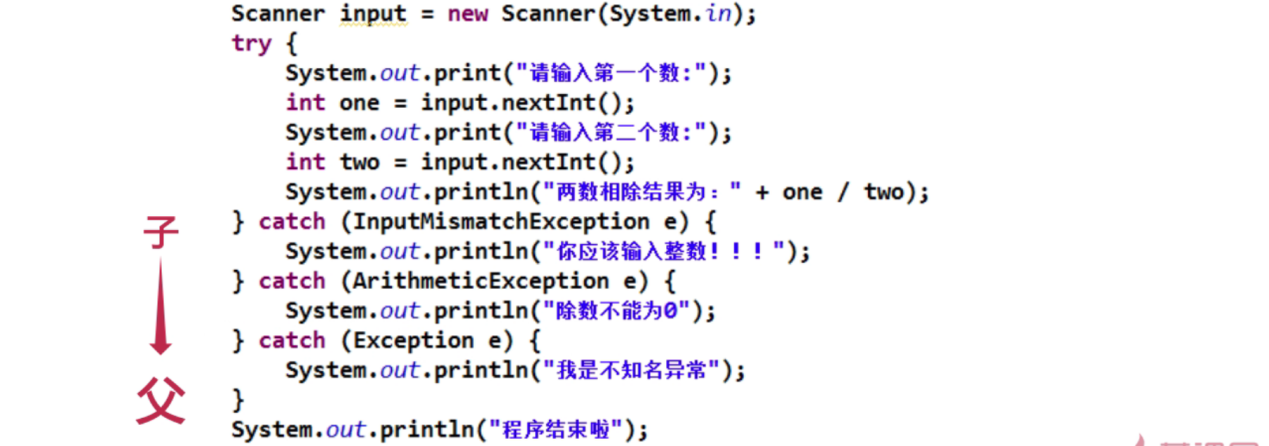
}finally{

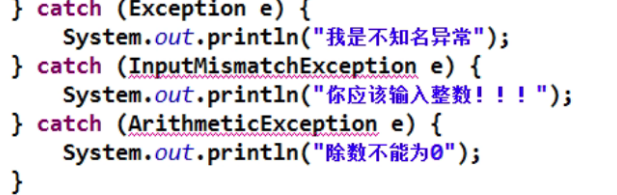
//最终将要执行的一些代码

}

由先子类后父类的顺序来编写多重catch语句块,因为异常处理器会就近寻找匹配的异常处理程序,而子类继承于父类,针对于父类的异常处理程序对于子类也是适用的.

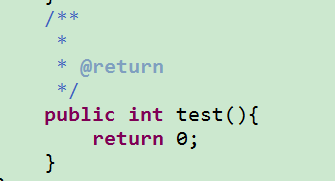
若不按此顺序则会报错.





**1-3 Java中通过案例学习try…catch…finally**

**//**在方法/属性上输入/\*\*后敲回车，Eclipse会自动安装模版里面的补充注释。

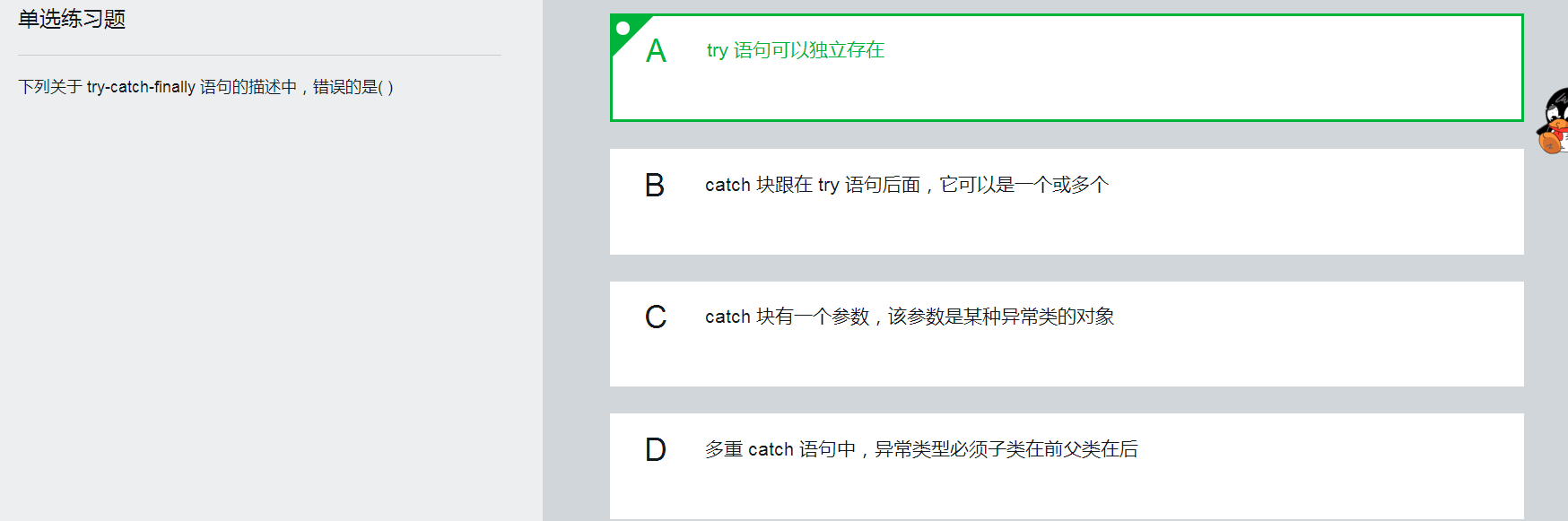
****

e.printStackTrace(); //打印异常位置,并输出在控制台console上

执行完finally后再返回返回值

在不终止VM的情况下，finally中的代码一定会执行。(System.exit(0))

**1-4 练习题**

****

**1-5 Java中的异常抛出以及自定义异常**

Java中的异常抛出

throw —— 将产生的异常抛出（动作）

程序在执行到throw语句时立即停止，它后面的语句都不执行。

throws —— 用于声明异常，例如，如果一个方法里面不想有任何的异常处理，则在没有任何代码进行异常处理的时候，必须对这个方法进行声明有可能产生的所有异常

public void 方法名（参数列表）

throws 异常列表{

//调用会抛出异常的方法或者：

throw new Exception（）;

}

每个异常之间用逗号隔开。

public Throwable initCause(Throwable cause)

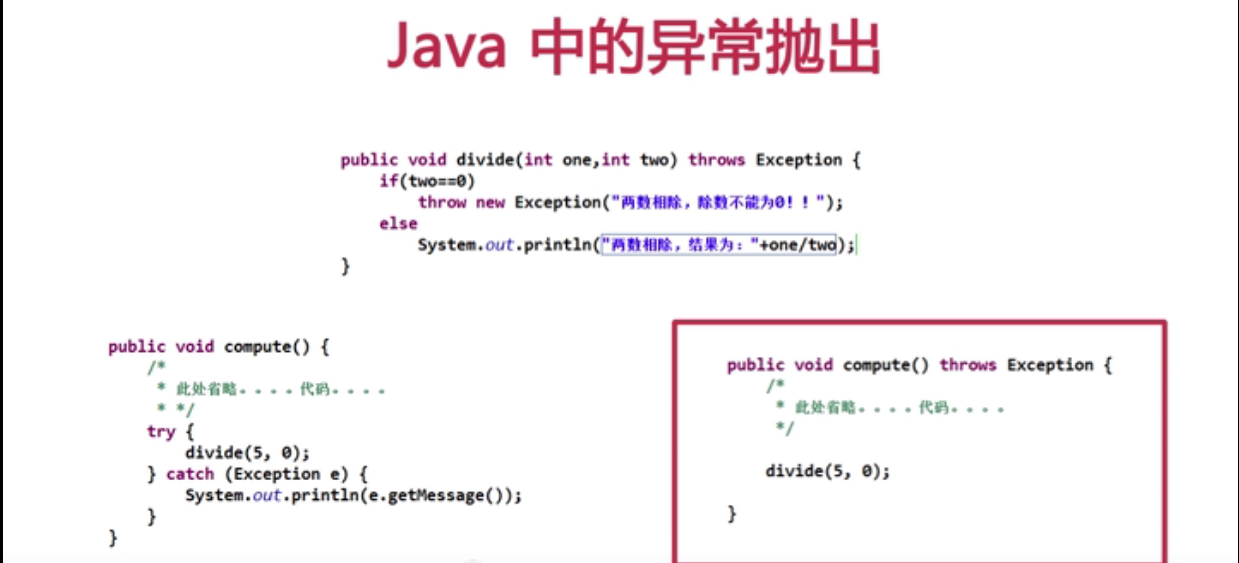
将此 throwable 的 cause 初始化为指定值。（该 Cause 是导致抛出此 throwable 的throwable。）

此方法至多可以调用一次。此方法通常从构造方法中调用，或者在创建 throwable 后立即调用。如果此 throwable 通过 Throwable(Throwable) 或 Throwable(String,Throwable) 创建，此方法甚至一次也不能调用。

**1-6 练习题**

**1-7 Java中的异常链**

把捕获的异常包装成新的异常，然后在新的异常里面添加对新的异常的引用，再把新异常抛出



自定义异常

class 自定义异常类 extends 异常类型{

}

**1-8 练习题**



**1-9 经验总结**

1. 处理运行时异常时，采取逻辑去合理规避同时辅助try-catch处理
2. 在多重catch块后面，可以加一个catch（Exception）来处理可能会被遗漏的异常
3. 对于不确定的代码，也可以加上try-catch，处理潜在的异常
4. 尽量去处理异常，切忌只是简单调用printStackTrace（）去打印输出
5. 具体如何处理异常，要根据不同的也无需求和异常类型去决定
6. 尽量添加finally语句块去释放占用的资源

**第二章** **认识Java中的字符串**

**2-1什么时候Java中的字符串**

在Java中,字符串被作为String类型的对象处理/String类位于java.lang包中，默认情况下，该包被自动导入所有的程序

创建String对象的方法：

String s1 = “imooc”;

String s2 = new String();

String s3 = new String(“imooc”);

**2-2 Java中字符串的不变性**

String对象创建后则不能被修改，是不可改变的，所谓的修改其实是创建了新的对象，所指向的内存空间不同.采用new 创建的字符串对象不进入字符串池。如下所示：

String s1 = “爱慕课”;

String s2 = “爱慕课”;

String s3 = new String(“爱慕课”);

String s4 = new String(“爱慕课”);

//多次出现的字符常量，Java编译程序只创建一个，所以返回true

System.out.println(s1 == s2);

//s1和s3是不同的对象，所以返回faulse

System.out.println(s1 == s3);

// s3和s4是不同的对象，所以返回faulse

System.out.println(s3 == s4);

//void

s1 = “欢迎来到：” + s1;

//字符串s1被修改，指向新的内存空间

System.out.println(s1);

运行结果：

true

false

false

欢迎来到：爱慕课

一旦一个字符串在内存中创建,则这个字符串将不可改变.如果需要一个可以改变的字符串,我们可以使用StringBuffer或者StringBuilder(后面章节会讲到).

每次new一个字符串就是产生一个新的对象,即便两个字符串的内容相同,使用”==”比较时也为”faulse”,如果秩序比较内容是否相同,应使用”equals()”方法.

“==”比较的是对象在内存中的地址值.

**2-3 Java中的String类的常用方法Ⅰ**

String类提供了许多用来处理字符串的方法，例如，获取字符串长度、对字符串长度、对字符串进行截取、将字符串转换为大写或小写、字符串分割等。

String类的常用方法：



String对象已经重写了equals方法,未重写时比较的是地址

== 基本类型比较的是地址，引用类型比较的是值

当调用 intern 方法时，如果池已经包含一个等于此 String 对象的字符串（该对象由 equals(Object) 方法确定），则返回池中的字符串。否则，将此 String 对象添加到池中，并且返回此 String 对象的引用。如果其中含有变量（String a=”1”; String b=a+”2”; ）则不会进入字符串池中。s1=s1.intern();

**2-4 Java中的String类的常用方法Ⅱ**

字节是计算机存储信息的基本单位，**1 个字节等于 8 位**， **gbk 编码中 1 个汉字字符存储需要 2 个字节**，**1 个英文字符存储需要 1 个字节**。所以我们看到上面的程序运行结果中，每个汉字对应两个字节值，如“学”对应 “-47 -89” ，而英文字母 “J” 对应 “74” 。同时，我们还发现汉字对应的字节值为负数，原因在于每个字节是 8 位，最大值不能超过 127，而**汉字转换为字节后超过 127，如果超过就会溢出，以负数的形式显示。**

**2-5 认识Java中的StringBuilder类**

在Java中,除了可以使用String类来存储字符串,还可以使用StringBuilder类或StringBuffer类存储字符串,那么它们之间有什么区别呢?

String类具有不可变性.如:

String str = “hello”;

System.out.println(str+”world”);

System.out.println(str);

从运行结果中我们可以看到，程序运行时会额外创建一个对象，保存·“helloworld”。当频繁操作字符串时，就会额外产生很多临时变量。使用StringBuilder则没有实现线程安全功能。所以性能略高。因此一般情况下，如果需要创建一个内容可变的字符串对象，应优先考虑使用StringBuilder类

定义StringBuilder类对象:

StringBuilder str1 = new StringBuilder();//创建一个空的StringBuilder对象

StringBuilder str2 = new StringBulder(“imooc”);//创建一个字符串”imooc”

System.out.println(str2);

运行结果:

imooc

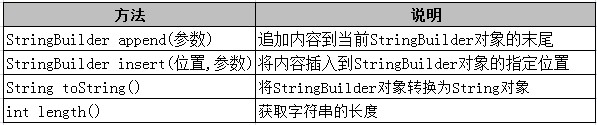
**String 字符串常量  
StringBuffer 字符串变量（线程安全）  
StringBuilder 字符串变量（非线程安全）**  
 简要的说， String 类型和 StringBuffer 类型的主要性能区别其实在于 String 是不可变的对象, 因此在每次对 String 类型进行改变的时候其实都等同于生成了一个新的 String 对象，然后将指针指向新的 String 对象，所以经常改变内容的字符串最好不要用 String ，因为每次生成对象都会对系统性能产生影响，特别当内存中无引用对象多了以后， JVM 的 GC 就会开始工作，那速度是一定会相当慢的。  
 而如果是使用 StringBuffer 类则结果就不一样了，每次结果都会对 StringBuffer 对象本身进行操作，而不是生成新的对象，再改变对象引用。所以在一般情况下我们推荐使用 StringBuffer ，特别是字符串对象经常改变的情况下。而在某些特别情况下， String 对象的字符串拼接其实是被 JVM 解释成了 StringBuffer 对象的拼接，所以这些时候 String 对象的速度并不会比 StringBuffer 对象慢，而特别是以下的字符串对象生成中， String 效率是远要比 StringBuffer 快的：  
 String S1 = “This is only a” + “ simple” + “ test”;  
 StringBuffer Sb = new StringBuilder(“This is only a”).append(“ simple”).append(“ test”);  
 你会很惊讶的发现，生成 String S1 对象的速度简直太快了，而这个时候 StringBuffer 居然速度上根本一点都不占优势。其实这是 JVM 的一个把戏，在 JVM 眼里，这个  
 String S1 = “This is only a” + “ simple” + “test”; 其实就是：  
 String S1 = “This is only a simple test”; 所以当然不需要太多的时间了。但大家这里要注意的是，如果你的字符串是来自另外的 String 对象的话，速度就没那么快了，譬如：  
String S2 = “This is only a”;  
String S3 = “ simple”;  
String S4 = “ test”;  
String S1 = S2 +S3 + S4;  
这时候 JVM 会规规矩矩的按照原来的方式去做

在大部分情况下 StringBuffer > String  
**StringBuffer**  
Java.lang.StringBuffer线程安全的可变字符序列。一个类似于 String 的字符串缓冲区，但不能修改。虽然在任意时间点上它都包含某种特定的字符序列，但通过某些方法调用可以改变该序列的长度和内容。  
可将字符串缓冲区安全地用于多个线程。可以在必要时对这些方法进行同步，因此任意特定实例上的所有操作就好像是以串行顺序发生的，该顺序与所涉及的每个线程进行的方法调用顺序一致。  
StringBuffer 上的主要操作是 append 和 insert 方法，可重载这些方法，以接受任意类型的数据。每个方法都能有效地将给定的数据转换成字符串，然后将该字符串的字符追加或插入到字符串缓冲区中。append 方法始终将这些字符添加到缓冲区的末端；而 insert 方法则在指定的点添加字符。  
例如，如果 z 引用一个当前内容是“start”的字符串缓冲区对象，则此方法调用 z.append("le") 会使字符串缓冲区包含“startle”，而 z.insert(4, "le") 将更改字符串缓冲区，使之包含“starlet”。  
在大部分情况下 StringBuilder > StringBuffer

**java.lang.StringBuilde**  
java.lang.StringBuilder一个可变的字符序列是5.0新增的。此类提供一个与 StringBuffer 兼容的 API，但不保证同步。该类被设计用作 StringBuffer 的一个简易替换，用在字符串缓冲区被单个线程使用的时候（这种情况很普遍）。如果可能，建议优先采用该类，因为在大多数实现中，它比 StringBuffer 要快。两者的方法基本相同。

对于三者使用的总结：1.如果要操作少量的数据用 = String  
　　　　　　 2.单线程操作字符串缓冲区 下操作大量数据 = StringBuilder  
　　　　　　 3.多线程操作字符串缓冲区 下操作大量数据 = StringBuffer

**2-6 Java中的StringBuilder类的常用方法**



**2-7 练习题**



**第三章 Java中必须了解的常用类**

**3-1 Java中的包装类**

int、float、double、boolean、char等基本数据类型是不具备对象的特性的，比如基本类型不能调用方法、功能简单。。。为了让基本数据类型也具备对象的特性，Java为每个基本数据类型都提供了一个包装类，这样我们就可以像操作对象那样来操作基本数据类型。

基本类型和包装类之间的对应关系：

基本类型 对应的包装类

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

Boolean Boolean

包装类主要提供了两大类方法：

1. 将本类型和其他基本类型进行转换的方法
2. 将字符串和本类型及包装类互相转换的方法

****

**3-2 Java中基本类型和包装类之间的转换**

基本类型和包装类之间经常需要互相转换，以Integer为例（其他几个包装类的操作相同）：

Integer a = new Integer(3);//定义Integer包装类对象，值为3

int b = a+5;//将对象和基本类型进行运算

在JDK1.5引入自动装箱和拆箱的机制后，包装类和基本类型之间的转换就更加轻松便利了。

装箱：把基本类型转换成包装类，使其具有对象的性质，又可分为手动装箱和自动装箱

int i = 10;//定义一个int基本类型值

Integer x = new Integer(i);//手动装箱

Integer y = i;//自动装箱

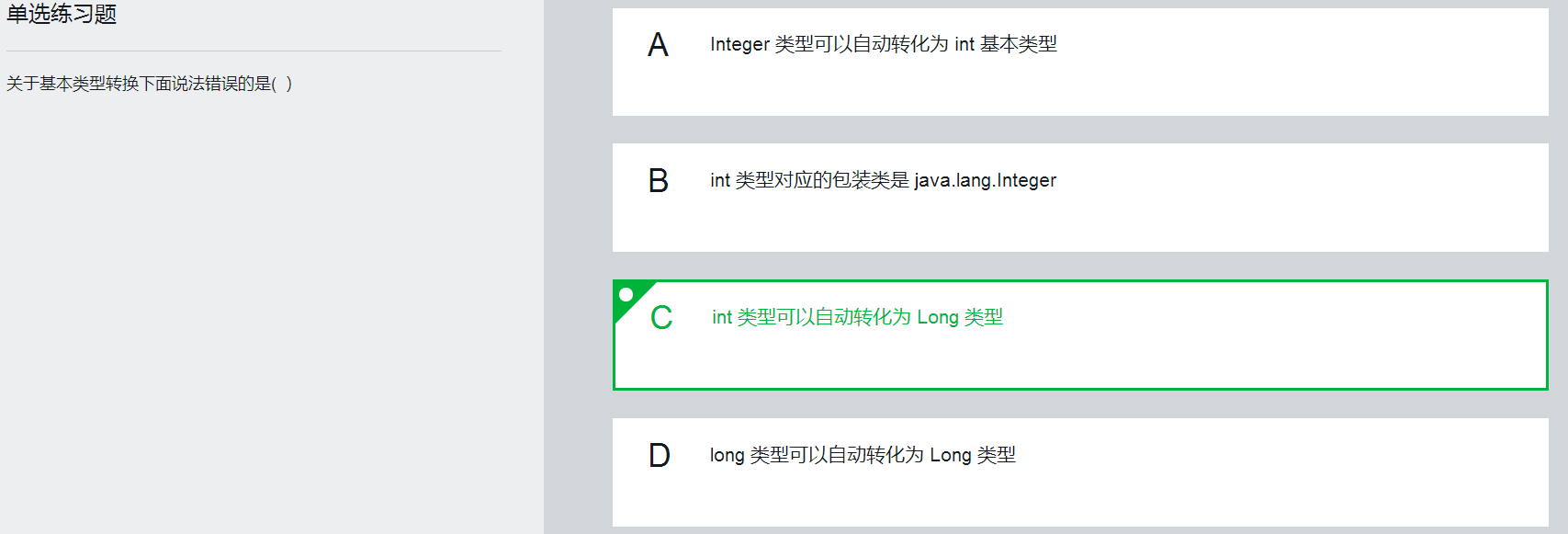
拆箱：和装箱相反，把包装类对象转换成基本类型的值，又可分为手动拆箱和自动拆箱

Integer j = new Integer(8);//定义一个Integer包装类对象，值为8

int m = j.intValue();//手动拆箱为int类型

int n = j;//自动拆箱为int类型

**3-3 练习题**

****

**3-4 Java中基本类型和字符串之间的转换**

将基本类型转换为字符串有三种方法:

1. 使用包装类的toString方法
2. 使用String类的valueOf方法
3. 用一个空字符串加上基本类型,得到的就是基本类型数据对应的字符串

int c = 10;

String str1 = Inetger.toString(c);//方法一

String str2 = String.valueOf(c);//方法二

String str3 = c+””;//方法三

将字符串转换成基本类型有两种方法：

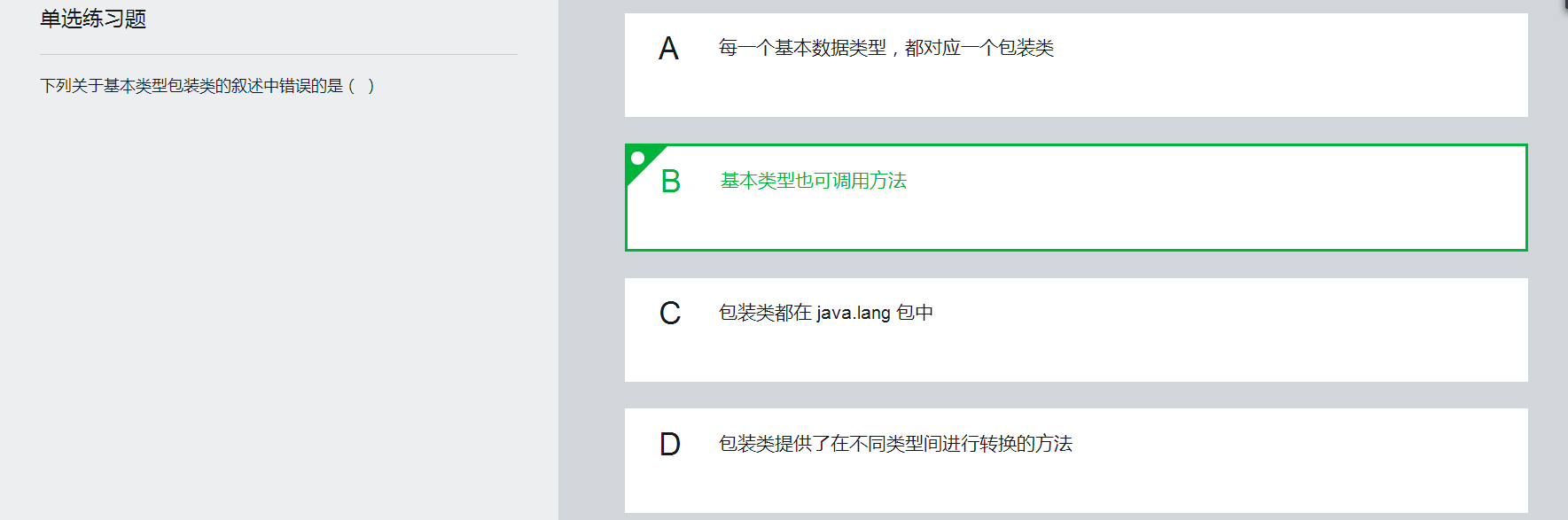
String str = “8”;

int d = Integer.parseInt(str);//方法一

int e = Integer.valueOf(str);//方法二

1. 调用包装类的parseXxx静态方法
2. 调用包装类的valueOf()方法转换为基本类型的包装类，会自动拆箱

**3-5 练习题**

****

**3-6 使用Date和SimpleDateFormat类表示时间**

在程序开发中，经常需要处理日期和时间的相关数据，此时我们可以使用java.util包中的Date类。这个类最主要的作用就是获取当前时间，我们来看下Date类的使用：

Date d = new Date();//使用默认的构造方法创建Date对象

System.out.println(d);//输出Date对象

使用Date类的默认无参构造方法常见出的对象就代表当前时间，我们可以直接输出Date对象显示当前的时间，显示的结果如下



其中，Wed代表Wednesday（星期三），Jun代表June（六月），11代表11号，CST代表China Standard Time（中国标准时间，也就是北京时间，东八区）

从上面的输出结果中，我们发现默认的时间格式不是很友好,与我们日常看到的日期格式不太一样,如果想要按指定的格式进行显示,如 2014-06-11, 那应该怎么做呢？

此时就到了java.text 包中的 SimpleDateFormat来对日期时间进行格式化，如可以将日期转换为指定格式的文本，也可以将日期转换为指定格式的文本，也可以将文本转换为日期。

1.使用format()方法将日期转换为指定格式的文本

//创建Date对象，表示当前时间

Date d = new Date();

//创建SimpleDateFormat对象，指定目标格式

SimpleDateFormat sdf = new SimpleDateFormat(“yyyy-MM-dd HH:mm:ss”);

//调用formatfangfa，格式化时间，转化为指定格式字符串

String today = sdf.format(d);

//输出转换后的字符串

System.out.println(today);

代码中的”yyyy-MM-dd HH:mm:ss”为预定义字符串，yyyy表示四位年，MM表示两位月份，dd表示两位日期，HH表示小时(使用24小时制)，mm表示分钟，ss表示秒这样就指定了转换的目标格式，最后调用format()方法将时间转换为指定的格式的字符串。

运行结果： 2014-06-11 09:55:48

2.使用parese()方法将文本转换为日期

//创建日期格式的字符串

String day = “2014年02月14日 10:30:25”

//创建SimpleDateFormat对象，指定字符串的日期格式

SimpleDateFormat df = new SimpleDateFormat(“yyyy年MM月dd日 HH:mmm:ss”);

//调用parse()方法，将字符串转换为日期

Date date = df.parse(day);

//输出转换后的时间

System.out.println(“当前时间”+date);

代码中的”yyyy年MM月dd日 HH:mm:ss”指定了字符串的日期格式，调用parse()方法将文本转换为日期。

运行结果：

当前时间:Fri Feb 14 10:30:25 CST 2014

注意：

1. 调用SimpleDateFormat对象的parse()方法时可能会出现转换异常，即ParseException，因此需要进行异常处理
2. 使用Date类时需要导入java.util包，使用SimpleDateFormat时需要导入java.text包

**参数表:**

字母 日期或时间元素 表示 示例

G Era 标识符 Text AD

y 年

字母 日期或时间元素 表示 示例   
G Era 标志符 Text AD   
y 年 Year 1996; 96   
M 年中的月份 Month July; Jul; 07   
w 年中的周数 Number 27   
W 月份中的周数 Number 2   
D 年中的天数 Number 189   
d 月份中的天数 Number 10   
F 月份中的星期 Number 2   
E 星期中的天数 Text Tuesday; Tue   
a Am/pm 标记 Text PM   
H 一天中的小时数（0-23） Number 0   
k 一天中的小时数（1-24） Number 24   
K am/pm 中的小时数（0-11） Number 0   
h am/pm 中的小时数（1-12） Number 12   
m 小时中的分钟数 Number 30   
s 分钟中的秒数 Number 55   
S 毫秒数 Number 978   
z 时区 General time zone Pacific Standard Time; PST; GMT-08:00   
Z 时区 RFC 822 time zone -0800

模式字母通常是重复的，其数量确定其精确表示：   
Text: 对于格式化来说，如果模式字母的数量大于或等于 4，则使用完全形式；否则，在可用的情况下使用短形式或缩写形式。对于分析来说，两种形式都是可接受的，与模式字母的数量无关。   
Number: 对于格式化来说，模式字母的数量是最小的数位，如果数位不够，则用 0 填充以达到此数量。对于分析来说，模式字母的数量被忽略，除非必须分开两个相邻字段。   
Year: 对于格式化来说，如果模式字母的数量为 2，则年份截取为 2 位数,否则将年份解释为 number。   
对于分析来说，如果模式字母的数量大于 2，则年份照字面意义进行解释，而不管数位是多少。因此使用模式 "MM/dd/yyyy"，将 "01/11/12" 分析为公元 12 年 1 月 11 日。   
  
在分析缩写年份模式（"y" 或 "yy"）时，SimpleDateFormat 必须相对于某个世纪来解释缩写的年份。这通过将日期调整为 SimpleDateFormat 实例创建之前的 80 年和之后 20 年范围内来完成。例如，在 "MM/dd/yy" 模式下，如果 SimpleDateFormat 实例是在 1997 年 1 月 1 日创建的，则字符串 "01/11/12" 将被解释为 2012 年 1 月 11 日，而字符串 "05/04/64" 将被解释为 1964 年 5 月 4 日。在分析时，只有恰好由两位数字组成的字符串（如 Character.isDigit(char) 所定义的）被分析为默认的世纪。其他任何数字字符串将照字面意义进行解释，例如单数字字符串，3 个或更多数字组成的字符串，或者不都是数字的两位数字字符串（例如"-1"）。因此，在相同的模式下， "01/02/3" 或 "01/02/003" 解释为公元 3 年 1 月 2 日。同样，"01/02/-3" 分析为公元前 4 年 1 月 2 日。   
  
Month: 如果模式字母的数量为 3 或大于 3，则将月份解释为 text；否则解释为 number。   
General time zone: 如果时区有名称，则将它们解释为 text。对于表示 GMT 偏移值的时区，使用以下语法：   
     GMTOffsetTimeZone:   
             GMT Sign Hours : Minutes   
     Sign: one of   
             + -   
     Hours:   
             Digit   
             Digit Digit   
     Minutes:   
             Digit Digit   
     Digit: one of   
             0 1 2 3 4 5 6 7 8 9   
Hours 必须在 0 到 23 之间，Minutes 必须在 00 到 59 之间。格式是与语言环境无关的，并且数字必须取自 Unicode 标准的 Basic Latin 块。   
对于分析来说，RFC 822 time zones 也是可接受的。   
  
RFC 822 time zone: 对于格式化来说，使用 RFC 822 4-digit 时区格式：   
     RFC822TimeZone:   
             Sign TwoDigitHours Minutes   
     TwoDigitHours:   
             Digit Digit   
TwoDigitHours 必须在 00 和 23 之间。其他定义请参阅 general time zones。   
对于分析来说，general time zones 也是可接受的。   
  
SimpleDateFormat 还支持本地化日期和时间模式 字符串。在这些字符串中，以上所述的模式字母可以用其他与语言环境有关的模式字母来替换。SimpleDateFormat 不处理除模式字母之外的文本本地化；而由类的客户端来处理。

**3-7 Calendar类的应用**

Date类最主要的作用就是获得当前时间,同时这个类里面也具有设置时间以及一些其他的功能,但是由于本身设计的问题,这些方法却遭到众多批评,不建议使用,更推荐使用Canlendar类进行时间和日期的处理.

java.util.Calendar类是一个抽象类,可以通过调用getInstance()静态方法获取一个Canlendar对象，此对象已由当前日期时间初始化，即默认代表当前时间，如Calendar c = Calendar.getInstance();

那么如何使用Calendar获取年、月、日、时间等信息呢？我们来看下面的代码：

Calendar c = Calendar.getInstance(); //创建Calendar对象

int year = c.get(Calendar.YEAR); //获取年

int month = c,get(Calendar.MONTH)+1; //获取月份，0表示一月份

int day = c.get(Calendar.DAY\_OF\_MONTH); //获取日期

int hour = c.get(Calendar.HOUR\_OF\_DAY); //获取小时

int minute = c.get(Canlendar.MINUTE); //获取分钟

int second = c.get(Canlendar.SECOND); //获取秒

System.out.println(“当前时间:”+year+”-”+month+” ”+day+” ”+hour+”:”+minute+”:”+second);

其中，调用Calendar类的getInstance()方法获取一个实例，然后通过调用get()方法获取日期时间信息，参数为需要获得的字段的值，Calendar.Year等为Calendar类中定义的静态常量。

运行结果：

当前时间: 2014-6-11 11:13:50

Calendar类提供了getTime()方法，用拉力获取Date对象，完成Calendar和Date的转换，还可以通过getTimeInMillis()fangfa1,获取此Calendar的时间值以毫秒为单位。如下所示：

Date date = c.getTime(); //将Calendar对象转换为Date对象

Long time = c.getTimeInMillis(); //获取当前毫秒数

System.out.println(“当前时间:”+date);

System.out.println(“当前毫秒数”+time);

运行结果:

当前时间: Wed Jun 11 11:26:59 CST 2014

当前毫秒数: 1402457219381

**3-8 使用Math类操作数据**

Math类位于java.lang包中，包含用于执行基本数学运算的方法，Math类的所有方法都是静态方法，所以使用该类中的方法时，可以直接使用类名.方法名，如: Math.round();

常用的方法：

返回值 方法名 解释

long round() 返回四舍五入后的整数

double floor() 返回小于参数的最大整数

double ceil() 返回大于参数的最小整数

double random() 返回[0,1)之间的随机浮点数

通过案例我们来认识一下他们的使用吧

double a = 12.81; //定义一个double类型变量

int b = (int)a; //将double类型强制转化为int类型，去掉小数位

System.out.println(“强制类型转换：”+b);

long c = Math.round(a); //调用round方法，进行四舍五入

System.out.println(“四舍五入：”+c);

double d = Math.floor(a); //调用floor方法，返回小于参数的最大整数

System.out.println(“floor： ”+d);

double e = Math.ceil(a); //调用ceil方法，返回大于参数的最小整数

System.out.println(“ceil：”+e);

double x = Math.random(); //调用ransom方法，产生[0,1)之间的随机浮点数

System.out.println(“随机数：”+x);

int y = (int)(Math.random() \* 99); //产生[0.99)之间的随机整数

System.out.println(“产生[0,99)之间的随机整数：”+y);

运行结果：

强制类型转换：12

四舍五入：13

floor：12.0

ceil：13.0

随机数： 0.42585849456602687

产生[0,99)之间的随机整数：30

**第四章 Java中的集合框架（上）**

**4-1 Java中的集合框架概述**

集合概述

集合的概念:  
·现实生活中:很多的事物凑在一起

·数学中的集合：具有共同属性的事物的总体

·Java中的集合类：是一种工具类，就像是容器，储存任意数量的具有共同属性的对象

集合的作用：

·在类的内部对数据进行组织

·简单而快速的搜索大数量的条目

·有的集合接口，提供了一系列排列有序的元素，并且可以在序列中间快速的插入或者删除有关元素

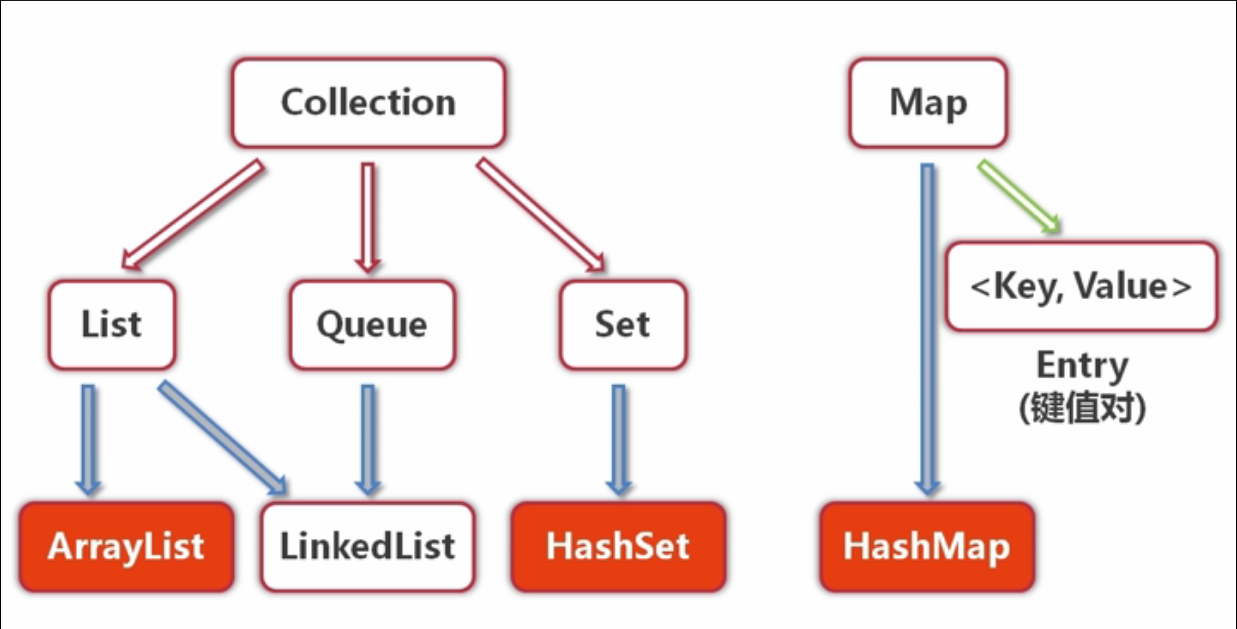
·有的集合接口，提供了映射关系，可以通过关键字（key）去快速查找到对应的唯一对象

与数组的对比—为何选择集合而不是数组:

·数组的长度固定,集合长度可变

·数组只能通过下表访问元素,类型固定,而有的集合可以通过任意类型查找所映射的具体对象

Java集合框架体系结构

****

List和Queue是有序且可以重复的,Set是无序且不可重复的

在每一个Collection类中,存储的是一个一个独立的对象。而Map类的内部会以<key,Value>作为映射存储数据，一个映射就是一个Entry类的实例（Entry类是Map类的内部类，翻译成键值对），Key和Value可以是任意格式的对象

**4-2 Collection接口 & List接口简介**

Collection接口、子接口以及实现类

Collection接口

·是List、Set和Queue接口的父接口

·定义了可用于操作List、Set和Queue的方法——增删改查

List接口及其实现类——ArrayList

·List是元素有序并且可以重复的集合，被称为序列

·List可以精确地控制每个元素的插入位置，或删除某个位置元素

·ArrayList——数组序列，是List的一个重要实现类，可以说是使用频率最高的一个

·ArrayList底层是由数组实现的

实现功能——模拟学生选课功能

·选择课程（往集合中添加课程）

·删除所选的某门课程（删除集合中的元素）

·查看所选课程

·修改所选课程

**4-3 学生选课—创建学生类和课程类**

**4-4 学生选课---添加课程Ⅰ**

List取元素：Course temp = （Course）couresesToSelect.get(0); //插入的元素cr1序号为0//对象存入集合都变成Object类型，取出时需要类型转换

List跟数组一样起始位置为0

当一个对象被添加进集合中去的时候，是会忽略他本来的类型的，只把他当做一个Object类型，取出时需要类型转换

//创建一个课程对象，并通过调用add方法，添加到备选课程List中

Course cr1 = **new** Course("1","数据结构");

coursesToSelect.add(cr1);

//第二种add方法

Course cr2 = **new** Course("2","C语言");

coursesToSelect.add(0, cr2);//会把原来0位置上的cr1挤到1位置上去

// //调用add方法时传入进的参数大于目前的长度 ,会抛出数组下标越界异常

// Course cr3 = new Course("2","test");

// coursesToSelect.add(3, cr3);

**4-5 学生选课---添加课程Ⅱ**

//另外两种添加课程的方法：

//方法一

Course[] course = {**new** Course("3","离散数学"),**new** Course("4","汇编语言")};

coursesToSelect.addAll(Arrays.*asList*(course));//把courses数组转化为List并传递进去

//目前: 0. 2 C语言 1. 1 数据结构 2. 3 离散数学 3. 4 汇编语言

//方法二

Course[] course2 = {**new** Course("5","高等数学"),**new** Course("6","大学英语")};

coursesToSelect.addAll(2,Arrays.*asList*(course2));

//目前: 0. 2 C语言 1. 1 数据结构 2. 5 高等数学 3. 6 大学英语

//4. 3 离散数学 5. 4 汇编语言

**4-6 学生选课—课程查询**

****

/\*\*

\* 取得List中的元素的方法

\* **@param** args

\*/

**public** **void** testGet(){

**int** size = coursesToSelect.size();//求List的长度

System.*out*.println("有如下课程待选:");

**for**(**int** i=0;i<size;i++){

**Course cr = (Course)coursesToSelect.get(i);//循环体中可重复**

System.*out*.println("课程:"+cr.id+":"+cr.name);

}

List中的元素可以重复的

****

**public** **void** testIterator(){

//通过集合的Iterator方法，取得迭代器的实例

Iterator it = coursesToSelect.iterator();//迭代器本身也是一个接口

System.*out*.println("有如下课程待选(通过迭代器访问):");

**while**(it.hasNext()){

Course cr = (Course)it.next();

System.*out*.println("课程:"+cr.id+":"+cr.name);

}

}

/\*\*

\* 通过for each方法访问集合元素

\*/

**public** **void** testForEach() {

**for**(Object obj:coursesToSelect){

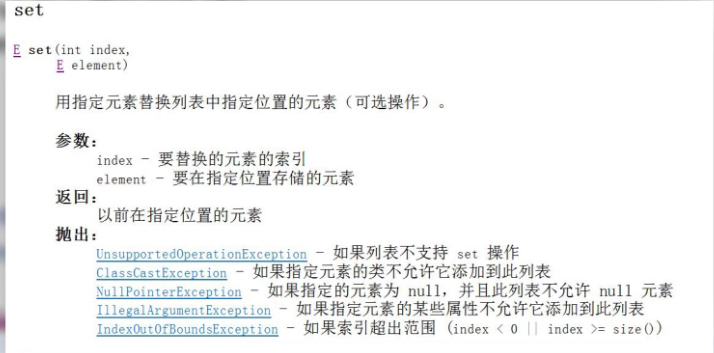
Course cr = (Course)obj;

System.*out*.println("课程:"+cr.id+":"+cr.name);

}

}

**4-7 学生选课—课程修改**

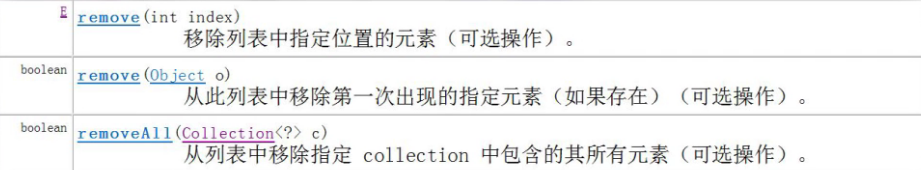
****

**public** **void** testModify(){

coursesToSelect.set(4,**new** Course("7","毛概"));

}

**4-8 学生选课—课程删除**

****

**public** **void** testRemove(){

Course cr =(Course)coursesToSelect.get(4);

System.*out*.println("我是课程:"+cr.id+":"+cr.name+",我即将被删除");

coursesToSelect.remove(cr);

System.*out*.println("成功删除课程");

}

**public** **void** testRemove2(){

System.*out*.println("即将删除4位置上的课程");

coursesToSelect.remove(4);

System.*out*.println("成功删除课程");

testForEach();

}

**public** **void** testRemove(){

System.*out*.println("即将删除4位置和5位置上的课程！");

Course[] courses={(Course)coursesToSelect.get(4),(Course)coursesToSelect.get(5)};

coursesToSelect.removeAll(Arrays.*asList*(courses));

System.*out*.println("成功删除课程");

testForEach();

}

**4-9 学生选课---应用泛型管理课程Ⅰ**

泛型

集合中的元素与,可以是任意类型的对象(对象的引用)

如果把某个对象放入鸡和,则会忽略他的类型,二把他当做Object处理

泛型则是规定了摸个集合只可以存放特定类型的对象

会在编译期间进行类型检查

可以直接按指定类型获取集合元素

**public** **class** TestGeneric {

/\*\*

\* 带有泛型——Course的List类型属性

\*/

**public** List<Course> courses;

**public** TestGeneric(){

**this**.courses=**new** ArrayList<Course>();

}

}

**4-10 学生选课---应用泛型管理课程Ⅱ**

泛型的集合不但可以存泛型类型的实例,还可以存泛型的子类型的实例

泛型集合中的限定类型不能使用基本数据类型

可以通过使用包装类限定限定允许存入的基本数据类型

**4-11 学生选课---通过Set集合管理课程**

ctrl+/快速注释

Set接口及其实现类----HashSet

·Set是元素无序并且不可以重复的集合,被称为集

·HashSet----哈希集,是Set的一个重要实现类

**第五章 Java中的集合框架（中）**

**5-1 Map & HashMap 简介**

Map接口

-Map提供了一种映射关系，其中的元素是以键值对（key-value）的形式存储的，能够实现根据key快速查找value

-Map中的键值对以Entry类型的对象实例形式存在

-键（key值）不可重复，value值可以

-每个键最多只能映射到一个值

-Map接口提供了分别返回key值集合、value集合以及Entry（键值对）集合的方法

-Map支持泛型，形式如:Map<K,V>

HashMap类

-HashMap是Map的一个重要实现类，也是最常用的，基于哈希表实现

-HashMap中的Entry对象是无序排列的

-Key值和value值都可以为null，但是一个HashMap只能有一个key值为null的映射

**5-2 学生选课---使用Map添加学生**

ctrl+shift+o 快速导入包

**5-3 学生选课---删除Map中的学生**

**5-4 学生选课---修改Map中的学生**

**第六章 Java中的集合框架（下）**

**6-1 学生选课---判断List中课程是否存在**

contains方法是通过调用equals方法判断的,通过修改equals方法修改contains方法

**6-2 学生选课---判断Set中课程是否存在**

**6-3 学生选课---获取List中课程的位置**

**6-4 学生选课---判断Map中是否包含指定的key和value**

**6-5 应用Collections.sort() 实现List排序**

**6-6 学生选课---尝试对学生序列排序**

**6-7 Comparable & Comparator**

**6-8 学生选课---实现学生序列排序**

**第七章 综合实战**

**7-1 建议扑克牌游戏**