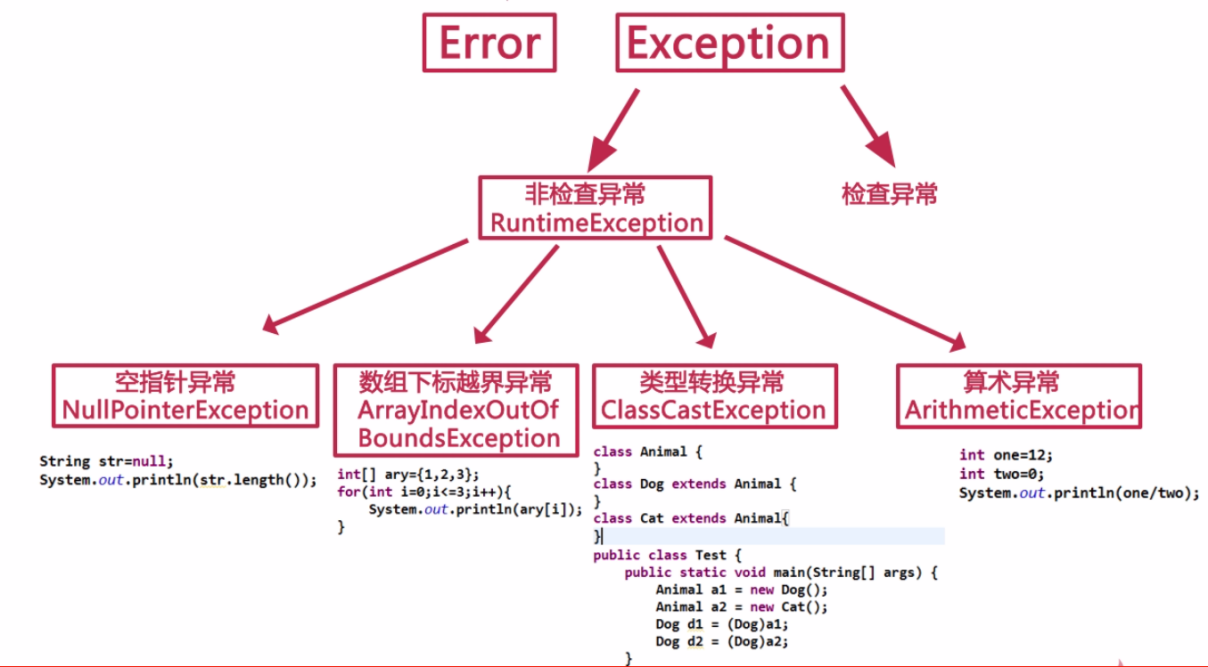
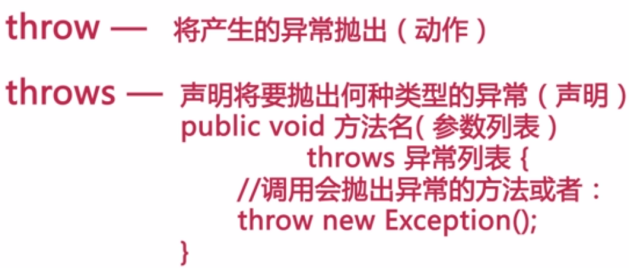
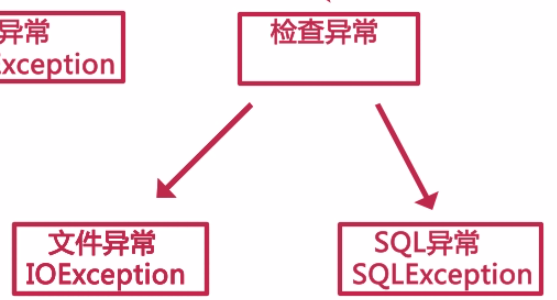
**一．异常与异常处理 根接口-Throwable**







1.**try**{

}

}**catch**(Exception e){ Excpetion e 是创建异常对象

e.printStackTrace();

}

1. 无try,catch 执行语句中throw 异常

if(true){

throw new Exception(“xxxxxxx”);

}

3.throws

若try块中的方法抛出异常，首先抛出异常的方法中止执行，程序的控制权交给catch块中的异常处理程序。

多重catch块出现异常就近寻找匹配的catch块，所以大catch放在后面

\*\*\*e.printStackTrace();打印堆栈错误信息

Catch块执行完毕之后，需要善后工作，比如关闭链接，关闭文件···

此时使用finally语句块

\*\*\*总结：

1.try...catch...finally...return四个语句块中return语句的执行顺序。

try -> catch -> finally -> finally.return -> catch.return -> try.return -> return”

没有相应的阶段请忽略该阶段，任何阶段的return被执行，则整个语句块结束

2.finally是在return执行后再执行的

（此时并没有执行return，而是先把return的值保存起来，执行finally【无return】，返回的值都不会改变，任然是之前保存的值；若finally【有return】，则执行finally的return）

3.finally中最好不要包含return，否则程序会提前退出，返回值不是try或catch中保存的返回值。

4.不管有木有出现异常，finally块中代码都会执行；

5.在catch中再throw new RuntimeException的，是为了中断程序，因为runtime的异常会中断程序，不再运行下去

1. **Java中的字符串**
2. Java 中，字符串被作为 String 类型的对象处理。String 类位于 java.lang 包

字符串是不可变对象，一旦创建，内容不变，改变内容一定会创建新对象

2.字符串拼接符 + 规则： 从左向右

+号两边都为数字，则做加法运算

+号两边出现字符串，则自动将右边数据全部转换为字符串输出

3.对于final修饰的字符串，编译期直接进行了常量值替换，而对于非final字段则是在运行期进行赋值处理的。

4.JAVA 1.6之后，常量字符串的“+”操作，编译阶段直接会合成为一个字符串。

String str1 = "aa";

String str2 = new String("aa");

String str3 = str2.intern(); //str2.intern();返回的是池中"aa"的引用str1

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

System.out.println(str1==str3); //T

5.字面量赋值的方式和new的方式生成字符串有本质的区别：

字面量赋值创建字符串时，会优先在常量池中查找是否已经存在相同的常量值，

若存在，引用直接指向该常量值；因指向相同，所以引用地址相同。

若不存在，则在常量池中生成一个常量值，再将引用指向该常量值。

new创建字符串时，就直接在堆中生成一个字符串的对象

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 ”false” ,如果只需比较内容是否相同，应使用equals()方法，String 内部已经重写了equals()方法

6.元空间Metaspace：JDK8 HotSpot JVM使用本地内存来存储类元数据信息

JDK 1.7 以后，HotSpot 已将常量池从永久代【PermGen】转移到了堆的【元空间Metaspace】中，同时intern方法发生改变：还是会先查询常量池中是否有已经存在，如果存在，则返回常量池中的引用，这与之前没有区别。

区别在于，如果常量池找不到对应的字符串，则不会再将字符串拷贝到常量池，而是在常量池中生成一个对原字符串的引用。 所以在进行字面量赋值的时候，如果【常量池中的引用指向的字符串】和【字面量】一样，则直接返回常量池中的引用

String str = "aa";

final String FinalStr = "aa";

String str1 = "aabb";

String str2 = "aa"+"bb";

String str3 = str+"bb"; //实际使用StringBuilder.append完成，创建了新对象

String str4 = FinalStr+"bb";

String str5 = new String("aabb").intern(); //返回的是池中aabb的引用str1

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

System.out.println(str1==str3); //F

System.out.println(str1==str4); //T

System.out.println(str1==str5); //T

String st1 = "abc";

System.out.println("abc" == st1); //T

1. 字符串 str 中字符的索引从0开始，范围为0 到 str.length()-1

2. 使用 indexOf 进行字符或字符串查找时，如果匹配返回位置索引；如果没有匹配结果，返回 -1 。

3.可以从指定位置开始查找，返回第一次找到字符串的位置

Int Index = str.indexOf(“xxxxx”,3)

4.使用 substring(beginIndex , endIndex) 进行字符串截取时，前包后不包

**String类的方法：**左边为方法返回类型，调用：（str.右边）



**字符串转换为字符数组**

char[] c1 = s1.toCharArray();

**获取字符串中的字符**

char ch = s1.charAt(index); 返回当前字符串给定位置的字符

Boolean:startsWith和endsWith:检查是否以“xxxx”字符串开头或结尾

Boolean b = str.startsWith(“xxxx”);

数组转化为字符串：Arrays.toString(arr);返回String类型对象，

因为String类内已经重写了toString方法，所以不会输出哈希码，而是字面量。

1.其他数据类型转化为String：静态方法valueOf或者+

Int i = 10;

String str = String.valueOf(i);

2.String转化为基本数据类型【需使用包装类】

Integer.parselnt(String s) s转换成int

Double.parseDouble(String s) s转换成double

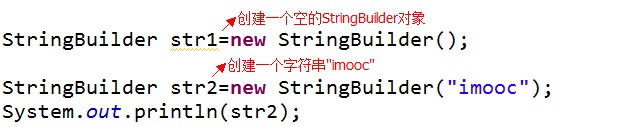
**Java中的StringBuilder类及使用方法**

String和StringBuilder之间的区别

String 类具有不可变性，改变内容会创建新的对象。StringBuilder可修改，还是原对象

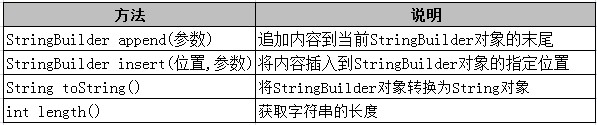
StringBuilder 和StringBuffer ，都在java.lang包下，不同之处，StringBuffer 是线程安全的，同步处理，性能慢，可将字符串缓冲区安全地用于多个线程

StringBuilder 则没有实现线程安全功能，不保证同步，所以性能略高。用在字符串缓冲区被单个线程使用的时候。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。



StringBuilder 内部已重写了toString（）方法，没有重写equals（）方法

**使用方法：**



append delete replace insert

boolean matches(String regex): 验证当前字符串是否满足正则表达式regex,无论正则表达式中是否有^...$都是做全匹配验证

String[] str.split(String regex)当前字符串中满足正则表达式的部分“切掉”，保留剩下的部分

String str.replaceAll(String regex,String replacement)字符串中满足正则表达式的部分替换为给定的字符串

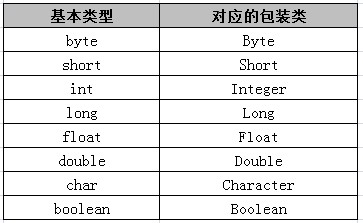
1. **Java中的包装类**

如 int、float、double、boolean、char 等。

基本数据类型不具备对象的特性，不能调用方法、功能简单

包装类可创建对象，调用方法，进行与基本类型和String类型的相互转换

基本类型和包装类之间的对应关系：



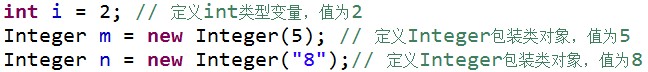
包装类主要提供了两大类方法：

1. 将本类型和其他基本类型进行转换的方法

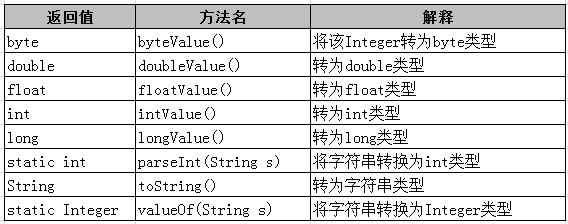
2. 将字符串和本类型及包装类互相转换的方法

Integer 包装类的构造方法：

IMG_256



**Integer包装类的常用方法： 对象.方法名（）**



6个数字类型的包装类都有两个常量

MAX\_VALUE 获取基本类型的最大值

MIN\_VALUE 获取基本类型的最小值

**基本类型和包装类之间的转换**

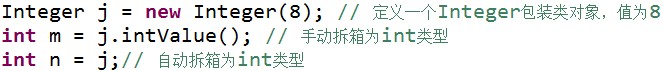
基本类型和包装类之间经常需要互相转换，以 Integer 为例（其他几个包装类的操作雷同）：

IMG_256

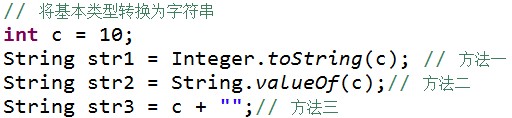
装箱：把基本类型转换成包装类，使其具有对象的性质，又可分为手动装箱和自动装箱



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



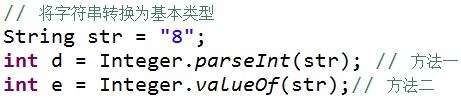
**基本类型和字符串之间的转换**



**字符串转换成基本类型有两种静态方法**

1. parseXxx（），前提是该字符串正确描述基本类型可以保存的值，否则会抛出异常

2. valueOf（），会自动拆箱



**使用Date和SimpleDateFormat类表示时间**

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：

IMG_256

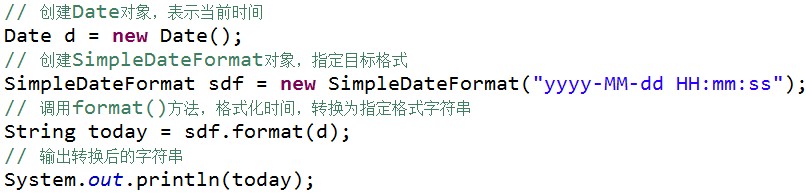
使用 Date 类的默认无参构造方法创建出的对象就代表当前时间，我们可以直接输出 Date 对象显示当前的时间，显示的结果如下：

IMG_256

按指定的格式进行显示，如 2014-06-11 09:22:30

 使用java.text 包中的 SimpleDateFormat 类

使用 format() 方法将日期转换为指定格式的文本



**运行结果： 2014-06-11  09:55:48**

long start = System.currentTimeMillis();获取1970.1.1至今的毫秒数

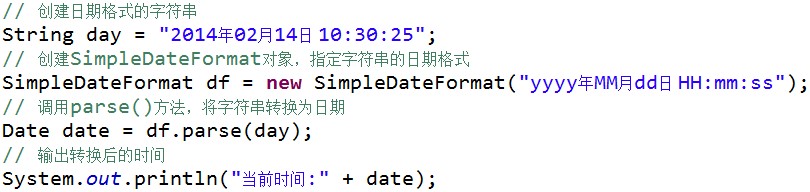
\* Date dt = new Date();

\* long ms = dt.getTime();//获取1970.1.1至dt的毫秒数

ms = ms + 1000\*60\*60\*24; //毫秒数+1天

dt.setTime(ms); //将毫秒数转换为时间

1. **使用 parse() 方法将文本转换为日期**



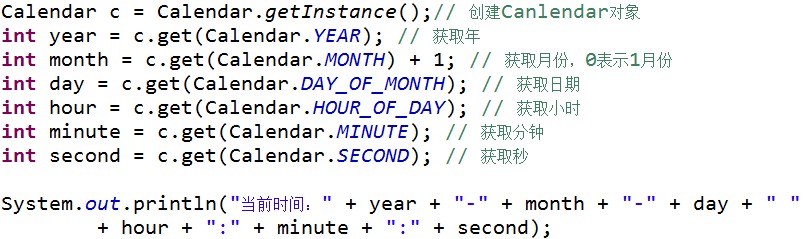
IMG_256

**Calendar类**

推荐使用 Calendar 类进行时间和日期的处理。Date不够灵活

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 Calendar 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()；但不可以直接输出c，需cl实例调用gettime()方法，返回值为Date类型的当前时间输出

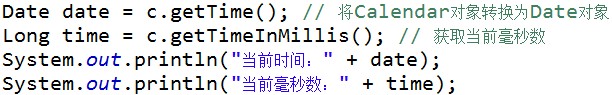
使用 Calendar 获取年、月、日、时间等信息：



Calendar.Year 等为 Calendar 类中定义的静态常量。

IMG_256

Calendar 类提供了 getTime() 方法，用来获取 Date 对象，完成 Calendar 和 Date 的转换，还可通过 getTimeInMillis() 方法，获取此 Calendar 的时间值，以毫秒为单位。



IMG_256

**Math类操作数据**

Math.pow(10, i) 10的i次方

**int** a=(**int**) (Math.*random*()\*100);

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();



**Scanner类输入数据**

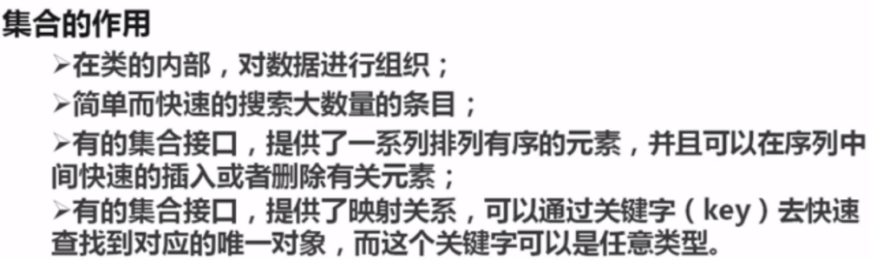
Scanner ipt = new Scanner

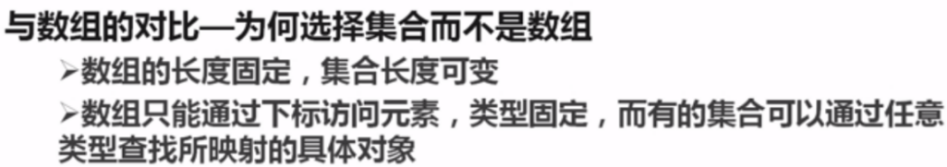
int age = ipt.nextInt();

double price = ipt.nextDouble();

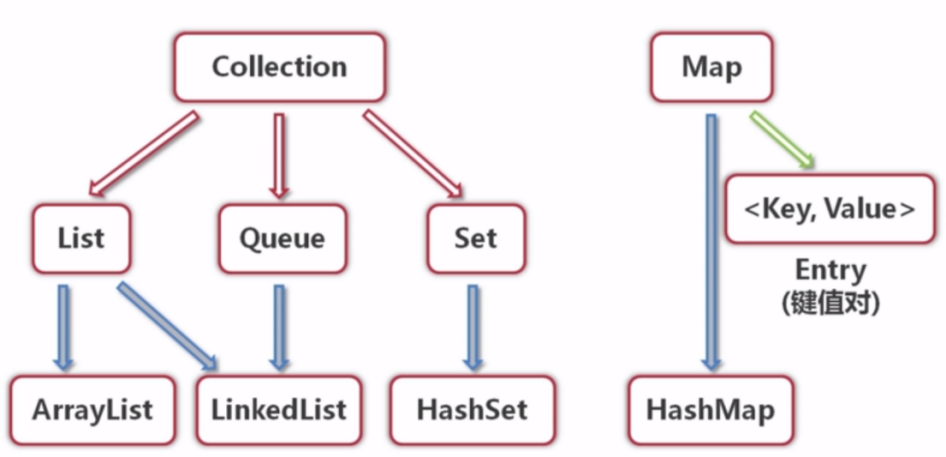
1. **Java中的集合框架**













**List(排列有序，可重复) - ArrayList(数组序列)**

不包含null，可手动添加

**1 .add/.get/.sort/.addAll/.size/.set修改/ .remove/ .isEmpty判断空/ .clear 清除/**

**.contains /.containsAll/.Iterator**

.subList获取指定范围内的子集，前包后不包

.subList获得的子集，属于原集合的一部分，相互影响

2.ArrayList便于查找。LinkedList便于删除，性能不同

3.将集合转换为数组Object[] courses=list.toArray();;

4.数组转换为集合arrays.asList，只能转换为List集合，转换相互影响

转换后的集合不能增删元素，且修改集合元素等于修改数组对应的元素

5.可以在创建当前集合的同时将给定的集合中的所有元素存入addAll，传入的集合无需与要创建的集合同类型,这也叫做复制构造器

**Set<String> set=new HashSet<String>(list);**  list传入set  
Set(无序，不可重复：集)HashSet(哈希集)

**add remove add Iterator**

没有提供get()set()的方法来获取修改集合中指定位置的元素，因为是无序的

Set中循环遍历每个对象，只能使用foreach或Iterator

Set的contains方法 该方法需求重写hashcode()和equals()方法

hashCode()方法的重写

原理：先调用每个元素的hashCode()方法返回哈希码，哈希码相等的情况下再调用equals方法进行比较

**Map:----<key,value>----HashMap(哈希表)**重要的实现类

**put remove entrySet keySet values**

HashTable是线程安全的，很多方法都是synchronized方法，

HashMap不是线程安全的，但其在单线程程序中的性能比HashTable要高

Map内部提供一种映射的关系，<key,value>两个对象为一个映射存储数据，这样一个映射就是Entry类的实例，Entry类是Map类的一个内部类，翻译为键值对，<key,value>为任意类型的对象，HashMap中的Entry对象是无序排列的，不可重复

1.key值不可以重复，value值可以重复， 二者都可以为null,但是一个HashMap只能有一个key值为null的映射,Map支持泛型，形式<K,V>

2 .put添加键值对 .get（key）获取对应的value get-put修改

.keySet该方法将当前map中的所有key存入一个set, 遍历set

.values遍历所有的value方法（很少用，不建议使用）

3.containsKey containsValue 都会调用equals方法

通过 .entrySet方法来遍历Map 获取key和value的键值对

Entry表示map内部的键值对，entry.getKey()/getValue()方法获取key和value

**Queue队列，FIFO先进先出**，有序，可重复，不支持迭代器，

pop()剔除第一个元素，front()返回第一个元素

**push pop add Iterator**

**add remove element 会抛出异常/ offer poll peek 返回特殊值null或false**

Queue队列数据结构扩展了Collection接口，除了基本的Collection操作，还提供了其他的插入、提取和检查操作

\* 优先使用自己的poll offer peek，优点是通过返回值判断成功与否

\* 队列通常以FIFO方式排序。

Queue实现通常不允许插入null元素，因为null也作用于poll方法的一个特殊的返回值，表明队列不包含元素。

LinkedList实现了该接口，原因是Queue经常要进行添加和删除操作，效率很高

**Deque双端队列，LIFO后进先出，**

是一个线性的collection，继承了队列接口queue，支持迭代器，有push\_back方法，**跟vector差不多**，比vector多了个pop\_front ,push\_front方法支持在两端插入和移除元素，只有有容量和没有容量

**addFirst()Last() remove get抛出异常 / offer(First)(Last) poll peek返回特殊值，不报异常**

**装载新元素push**

**用作LIFO堆栈**，应优先使用此接口而不是遗留stack类，使用时元素被推入双端队列的开头并从双端队列开头弹出。

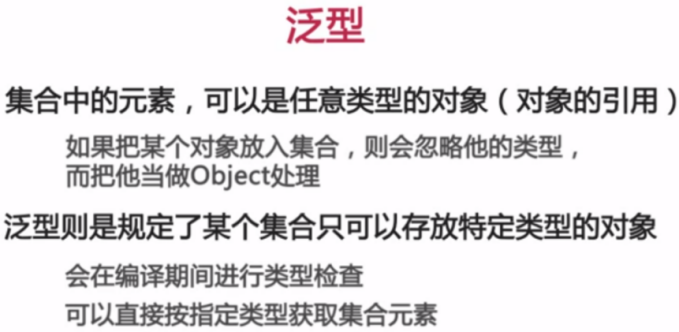
\* 堆栈方法完全等效于Deque方法

\* 如果将Deque限制为只能从一端入队和出队，则可实现栈(Stack)的数据结构。

\* 对于栈而言，入栈称之为push，出栈称之为pop.

\* 即从队列的两端分别可以入队offer和出队poll ， LinkedList实现了该接口

\* 双端队列指的是队列两端都可以进出队，但如果只从一端进出队，就实现了栈，提供的两个方法为：pop()和push()



泛型是一个参数化类型，不指定默认为object类型

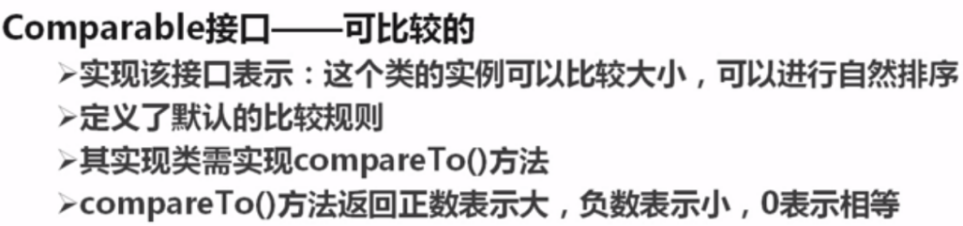
泛型可以修饰属性、方法的参数以及返回值的类型

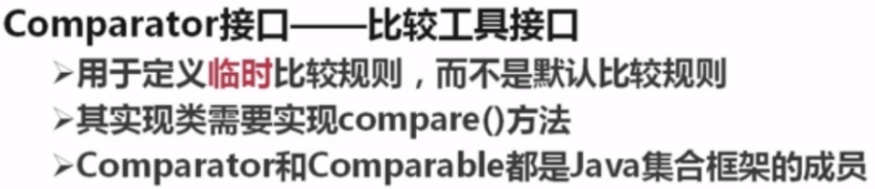
泛型集合可以添加泛型的子类型的对象实例

泛型不能使用基本类型，但可以使用包装类

\*\*如果在Java中给某个集合通过泛型规定其可存放元素的类型，添加的时候只可以添加某个特定类型的对象，取出的时候无需类型强转

**comparable的使用：Collections.sort(待排序序列);**





comparator的使用：Collections.sort(待排序序列,new Comparator对象名());

**通过Collections.sort()方法**

对Integer泛型的List进行排序

对String泛型的List进行排序，实现Comparable接口，可以进行默认规则的排序

比较每个字符串的 第0位字符 到第N位字符 ，优先比较id数字的左数第一位

\***默认规则**：先数字，后字母 0-9 A-Z a-z

\*如果是int型的，则进行比较数字大小，此处id是String类型

\*如果是String汉字的排序是按照Unicode执行的，优先比较汉字的左数第一位

\*String类无需实现comparable接口，String内部已经实现了