1、课程名称：Java 常用类库

2、知识点

2.1、本节预计讲解的知识点

1、 StringBuffer 类及其操作方法
2、 国际化程序的实现
3、 日期操作
4、 比较器

3、具体内容

3.1、StringBuffer 类（重点）

3.1.1、什么是 StringBuffer

从开发角度来看，在实际的操作中经常会出现字符串内容循环修改的情况，但是如果此时直接使用 String 做，则代码性能会非常的低，因为 String 的内容不可改变。

所以在 java 中提供了一个可以修改的字符串类，此类称为 StringBuffer。

在 String 中使用“+”号完成字符串的连接操作，那么在 StringBuffer 类中使用 append()方法完成字符串的连接操作，StringBuffer 类定义在 java.lang 包中。

3.1.2、StringBuffer 的常用方法

StringBuffer 本身是一个类，则类中会有很多的操作方法，操作方法如下：

No.	方法名称	类型	描述
1	public StringBuffer()	构造	构造一个空的 StringBuffer 对象
2	public StringBuffer(String str)	构造	将指定的 String 变为 StringBuffer 的内容
3	public StringBuffer(CharSequence seq)	构造	接收 CharSequence 接口的实例
4	public StringBuffer append(数据类型 b)	普通	提供了很多的 append()方法，用于进行字符串连接
5	public StringBuffer delete(int start,int end)	普通	删除指定位置的内容
6	public int indexOf(String str)	普通	字符串的查询功能
7	public StringBuffer insert(int offset,数据类型 b)	普通	在指定位置上增加一个内容
8	public StringBuffer replace(int start,int end,String str)	普通	将指定范围的内容替换成其他内容
9	public StringBuffer reverse()	普通	字符串反转
10	public String substring(int start)	普通	字符串截取

11	public String substring(int start,int end)	普通	截取指定范围的字符串

注意：

1、 String 本身已经实现了 CharSequence 接口，那么此接口的作用是进行字符的操作。

2、 既然所有的 append()方法都返回 StringBuffer 的对象，那么此时就可以一直使用 StringBuffer 的 append()方法进行字符串的连接，例如：StringBuffer 对象.append().append().append() ….

3、 StringBuffer 不能像 String 那样采用直接赋值的方式操作，必须使用构造方法的形式操作。

3.1.3、StringBuffer 的操作实例

3.1.3.1、字符串连接

在字符串的连接中使用的方法是 append()方法。

```
public class StringBufferDemo01 {
    public static void main(String[] args) {
        StringBuffer buf = new StringBuffer();
        buf.append("hello"); // 添加字符串
        buf.append("world").append("!!!\n").append("JAVA"); // 字符串连接
        System.out.println(buf) ;
    }
}
```

在以上的操作中所有的操作最终的返回结果是 StringBuffer，那么此时就不能够使用 String 类进行接收，如果要将一个 StringBuffer 类的对象变为 String 的内容，可以使用 StringBuffer 类中从 Object 覆写的 **toString**()方法完成。

```
public class StringBufferDemo02 {
    public static void main(String[] args) {
        StringBuffer buf = new StringBuffer();
        buf.append("hello"); // 添加字符串
        buf.append("world").append("!!!\n").append("JAVA"); // 字符串连接
        String str = buf.toString();// 将StringBuffer变为String
        System.out.println(str);
    }
}
```

3.1.3.2、字符串替换操作

在 StringBuffer 中提供了一个 replace()方法。可以直接将指定范围的内容进行替换。

```
public class StringBufferDemo03 {
    public static void main(String[] args) {
        StringBuffer buf = new StringBuffer();
        buf.append("hello"); // 添加字符串
        buf.append("world").append("!!!").append("JAVA"); // 字符串连接
        buf.replace(0, 5, "vince") ;
        System.out.println(buf) ;
    }
```

```
}
```

3.1.3.3、字符串反转操作

使用 reverse()方法完成字符串的反转操作。

```
public class StringBufferDemo04 {
    public static void main(String[] args) {
        StringBuffer buf = new StringBuffer();
        buf.append("hello"); // 添加字符串
        buf.append("world").append("!!!").append("JAVA"); // 字符串连接
        buf.reverse();// 字符串反转
        System.out.println(buf);
    }
}
```

3.1.3.4、字符串增加操作

StringBuffer 中最大的好处是可以在指定的位置上加入指定的字符串。使用一系列的 insert()方法完成。

```
public class StringBufferDemo05 {
    public static void main(String[] args) {
        StringBuffer buf = new StringBuffer();
        buf.append("world").append("!!!").append("JAVA"); // 字符串连接
        buf.insert(0, "hello ").insert(5, "~~~") ;
        System.out.println(buf);
    }
}
```

3.1.3.5、字符串截取操作

截取的方法是 substring()，此方法 String 类中也存在。

```
public class StringBufferDemo06 {
    public static void main(String[] args) {
        StringBuffer buf = new StringBuffer();
        buf.append("world").append("!!!").append("JAVA"); // 字符串连接
        buf.insert(0, "hello ").insert(5, "~~~") ;
        System.out.println(buf.substring(10));
        System.out.println(buf.substring(0,14));
    }
}
```

3.1.4、StringBuffer 的应用

StringBuffer 的应用只在于字符串的循环修改上，例如：如下的代码是一种错误的操作：

```
public class StringBufferDemo07 {
    public static void main(String[] args) {
        String str = "";
        for (int x = 0; x < 1000; x++) {
            str += x;   // 进行字符串修改
        }
        System.out.println(str);
    }
}
```

以上的操作性能绝对很低，那么对于这种情况，肯定要使用 StringBuffer 完成。如果使用 StringBuffer 则采用如下的形式：

```
public class StringBufferDemo08 {
    public static void main(String[] args) {
        StringBuffer str = new StringBuffer(); // StringBuffer必须通过构造方法实例
        for (int x = 0; x < 1000; x++) {
            str.append(x);  // 进行字符串修改
        }
        System.out.println(str);
    }
}
```

以上的操作结果的性能绝对要比之前的代码高很多，所以开发中只要是牵扯到循环修改数据的时候则肯定就一个原则，使用 StringBuffer。

例如：在搜索引擎上，如果现在要想搜索多个关键字，多个关键字之间要加上“ ”，例如：“java vince hello”。

既然 StringBuffer 本身是可以修改的，那么就可以将 StringBuffer 的对象传递的普通的方法之中完成引用传递。

```
public class StringBufferDemo09 {
    public static void main(String[] args) {
        StringBuffer str = new StringBuffer(); // StringBuffer必须通过构造方法实例
        str.append("hello ");
        fun(str); // 接收引用
        System.out.println(str);//hello world
    }
    public static void fun(StringBuffer s) {
        s.append("world");
    }
}
```

采用 String

```
package vince.stringbuffer;
public class StringBufferDemo01 {
    public static void main(String[] args) {
        String str="hello";
        fun(str); // 接收引用
        System.out.println(str);//hello
```

```
    }
    public static void fun(String s) {
        s=s+("world");
    }
}
```

3.2、国际化程序（理解）

国际化程序可以这样理解：

- 同一套程序代码可以在各个语言环境下进行使用。
- 各个语言环境下，只是语言显示的不同，那么具体的程序操作本身都是一样的，那么国际化程序完成的就是这样的一个功能。

在实际的开发中如果要想进行一个国际化程序的开发，则肯定会非常的麻烦，而且此种项目一般较少，那么理解国际化的基本操作和实现思想即可。

所谓的国际化的实现核心在于显示的语言上，那么对于这些显示的语言，最好的做法是将其定义成若干个属性文件（文件后缀是*.properties），属性文件中的格式采用“key=value”的形式进行操作。

如果要想实现国际化程序则肯定需要依靠以下几个类：

- java.util.Locale 类：用于表示一个国家的语言类
- java.util.ResourceBundle 类：用于读取资源文件（属性文件，*.properties）
- java.text.MessageFormat 类：用于进行文本的格式化操作

3.2.1、Locale 类

Locale 类本身表示一个国际化程序的语言操作类，是整个操作的核心，那么此类的操作方法如下：

No.	方法名称	类型	描述
1	public Locale(String language,String country)	构造	构造的时候跟上语言和国家
2	public static Locale getDefault()	普通	根据本地的语言环境得到 Locale 类对象

如果现在要想设置一个表示中文的 Locale 对象，则必须使用如下的方式：new Locale("zh","CN") ;

如果是英文的美国的语言环境：new Locale("en","US") ;

但是，以上的这些代码你是怎么知道的呢？

浏览器本身是不是属于多国语言的操作，那么实际上所有的国家的编码都可以通过浏览器找到。

【工具】 → 【Internet 选项】 → 【语言】

没有任何的必要去强记这些操作，因为所有的编码在浏览器之上全都有显示。

3.2.2、ResourceBundle 类

ResourceBundle 类表示的是一个资源文件的读取操作，所有的资源文件需要使用 ResourceBundle 进行读取，读取的时候不需要加上文件的后缀，文件的后缀默认就是“*.properties”。

那么此类的方法如下：

No.	方法名称	类型	描述
1	public static final ResourceBundle getBundle(String baseName)	普通	根据本地的语言环境，并设置要操作的属性文件的名称，得到实例
2	public static final ResourceBundle getBundle(String baseName,Locale locale)	普通	根据指定的语言环境，设置要操作属性文件的名称，得到实例
3	public final String getString(String key)	普通	根据资源文件中的 key 取得对应的 value

下面就使用 ResouceBundler 类得到资源文件中的内容。

1、 建立一个资源文件：Message.properties

```
info = Hello
```

2、 建立操作类，访问此属性文件，并且得到相关的信息

```
import java.util.ResourceBundle;
public class InternationDemo01 {
    public static void main(String[] args) {
        // 得到ResourceBundle类的实例，并且设置要操作的资源文件的名称
        ResourceBundle res = ResourceBundle.getBundle("Message");
        String str = res.getString("info"); // 根据所指定的资源文件中key取得与之对应value
        System.out.println("内容: " + str) ;
    }
}
```

那么以上的程序中可以发现通过 ResouceBundle 类找到了资源文件中的内容并进行显示。

3.2.3、实现国际化

下面编写一个程序，此程序可以显示中文的“你好”或者显示英文的“hello”，为了完成此种操作，那么此时需要建立多个资源文件，而且每一个资源文件的后缀是不同的，后缀的形式“资源文件_后缀.properties”/

1、 中文的资源文件：Message_zh_CN.properties

• 但是在 Eclipse 中在保存中文信息的时候将出现提示，说只能保存的是“ISO 8859-1 的英文”内容，而不能保存中文内容，所以，此时如果是想操作中文的话，则必须对中文进行转码操作，使用 JDK 目录中的一个“native2ascii.exe”。

```
# 你好！！！！！！
info = \u4f60\u597d\uff01\uff01\uff01\uff01\uff01\uff01\uff01
```

“#”开头的信息表示的是注释，只要是注释中也不能存在中文。

2、 英文的资源文件：Message_en_US.properties

```
info = Hello !!!!!!!!!
```

之后，编写程序，同时建立两个 Locale 对象，一个表示读取中文，另外一个表示读取英文。

```
import java.util.Locale;
import java.util.ResourceBundle;
public class InternationDemo02 {
    public static void main(String[] args) {
        Locale zhLoc = new Locale("zh", "CN"); // 得到中文的Locale对象
        Locale enLoc = new Locale("en", "US"); // 得到英文的Locale对象
        // 得到ResourceBundle类的实例，并且取得资源文件的名称
        // 以下得到的是中文的资源文件：Message_zh_CN.properties
        ResourceBundle zhRes = ResourceBundle.getBundle("Message", zhLoc);
        // 以下得到的是英文的资源文件：Message_en_US.properites
        ResourceBundle enRes = ResourceBundle.getBundle("Message", enLoc);
        System.out.println("中文内容：" + zhRes.getString("info"));
        System.out.println("英文内容：" + enRes.getString("info"));
    }
}
```

此时，只要设置了正确的区域，则会读取不同的文件，根据资源文件的所指定的区域。

因为此时存在了更家具体的语言的资源文件，所以“Message.properties”根本就不会被读取到。

如果此时要读的具体的资源文件不存在。则会使用默认的文件进行内容的读取。

3.2.4、处理动态文本

之前的所有操作中，读取的内容都是固定的，但是如果现在假设要想打印这样的信息“欢迎，Xxx 光临！”，具体的登陆名称不是固定的，所以之前的操作就会存在问题了，那么此时就可以使用动态文本进行程序的处理。

如果要想进行动态的文本处理，则必须使用 java.text.MessageFormat 类完成。此类是 java.text.Format 的子类。

java.text.Format 类中有三个常用的子类：MessageFormat、DateFormat、NumberFormat。

修改之前的资源文件：

1、 修改 Message_zh_CN.properties 文件：

```
# 你好，{0}，欢迎{1}光临！
info = \u4f60\u597d\uff0c{0}\uff0c\u6b22\u8fce{1}\u5149\u4e34\uff01
```

2、 修改 Message_en_US.properties 文件：

```
info = Hello ,{0}.Welcome {1} !
```

以上有两个参数，所以在使用的时候就需要准确的设置，使用的方法如下：

No.	方法名称	类型	描述
1	public static String format(String pattern,Object... arguments)	普通	设置动态文本，采用可变参数的形式进行设置

范例：读取动态文本

```
import java.text.MessageFormat;
import java.util.Locale;
import java.util.ResourceBundle;
public class InternationDemo03 {
    public static void main(String[] args) {
        Locale zhLoc = new Locale("zh", "CN"); // 得到中文的Locale对象
        Locale enLoc = new Locale("en", "US"); // 得到英文的Locale对象
        // 得到ResourceBundle类的实例，并且取得资源文件的名称
        // 以下得到的是中文的资源文件：Message_zh_CN.properties
        ResourceBundle zhRes = ResourceBundle.getBundle("Message", zhLoc);
        // 以下得到的是英文的资源文件：Message_en_US.properites
        ResourceBundle enRes = ResourceBundle.getBundle("Message", enLoc);
        String zhInfo = zhRes.getString("info"); // 读取中文的内容，但是里面存在占位符
        String enInfo = enRes.getString("info"); // 读取英文的内容，但是里面存在占位符
        System.out
                .println("中文内容：" + MessageFormat.format(zhInfo, "爪哇", "java"));
        System.out.println("英文内容："
                + MessageFormat.format(enInfo, "pig", "psir"));
    }
}
```

以上在开发中使用较多，属于动态文本的操作。

3.3、其他操作类

以上的操作都属于比较大的一些操作类，代码都比较多，但是在 Java 的类库中还有一些小的操作类，下面来观察：Math、Random、Arrays 类

3.3.1、Math 类

Math 类表示的是数学的计算操作，在 Math 类中提供了很多的操作方法。在此类中有一个方法很重要：

- 四舍五入：public static long round(double a)

```
public class MathDemo {
    public static void main(String[] args) {
        System.out.println("PI: " + Math.PI);
        System.out.println("ROUND: " + Math.round(234.567));
        System.out.println("ROUND: " + Math.round(234.567*100)/100.00);
    }
}
```

3.3.2、Random 类

java.util.Random 类的作用是生成随机数，操作方法如下：

No.	方法名称	类型	描述
1	public Random()	构造	实例化对象
2	public int nextInt(int n)	普通	取出不大于 n 的整数
3	public float nextFloat()	普通	取出小数

范例：使用 Random 类生成 10 个不大于 100 的整数

```
package org.randomdemo;
import java.util.Random;
public class RandomDemo {
    public static void main(String[] args) {
        Random r = new Random();
        for (int x = 0; x < 10; x++) {
            int temp = r.nextInt(100);
            System.out.print(temp + "、");
        }
    }
}
```

3.3.3、Arrays 类

java.util.Arrays 类在之前的数组排序的操作中已经讲解过了，实际上此类是一个专门操作数组的类，那么在此类中规定了以下的方法：

No.	方法名称	类型	描述
1	public static int binarySearch(数据类型[] a,数据类型 key)	普通	二分查找法，必须保证数组是排序的操作
2	public static boolean equals(数据类型[] a,数据类型[] a2)	普通	比较两个数组的内容是否相等
3	public static void fill(数据类型[] a,数据类型 val)	普通	以指定的内容对数组进行填充操作
4	public static void sort(数据类型[] a)	普通	数组的排序操作
5	public static String toString(数据类型[] a)	普通	输出数组

范例：观察 Arrays 类的使用

```
package org.randomdemo;
import java.util.Arrays;
public class ArraysDemo {
    public static void main(String[] args) {
        int x[] = { 1, 3, 4, 1, 43, 4, 5, 76, 7, 8, 9, 0, 1, 23, 6 }; // 数组无序
        System.out.println("声明的数组：" + Arrays.toString(x)); // 数组输出
        Arrays.sort(x); // 排序
        System.out.println("排序的数组：" + Arrays.toString(x)); // 数组输出
        System.out.println("二分查找法：" + Arrays.binarySearch(x, 5));
        Arrays.fill(x, 3); // 数组中的内容使用数字3填充
        System.out.println("填充的数组：" + Arrays.toString(x)); // 数组输出
```

```
    }
}
```

3.4、日期操作类（重点）

在所有的编程语言中都会对日期有所支持，在 Java 中也提供了很多的类可以对日期进行支持的。

3.4.1、Date 类

java.util.Date 类的使用非常简单。此类使用的时候直接输出即可。

```
package org.datedemo;
import java.util.Date;
public class DateDemo {
    public static void main(String[] args) {
        Date date = new Date(); // 实例化Date对象
        System.out.println(date);   // 输出Date对象
    }
}
```

程序运行效果：

```
Tue Mar 31 14:02:39 CST 2009
```

以上确实可以显示出时间，但是这样的时间格式并不完全符合于国人的喜好，所以，如果要想得到一个好的时间，则必须使用 Calendar 类，通过此类可以得到一个比较完整的时间。

3.4.2、Calendar

使用此类可以将时间精确到毫秒显示，此类的定义如下：

```
public abstract class Calendar
extends Object
implements Serializable, Cloneable, Comparable<Calendar>
```

此类是一个抽象类，既然是抽象类则使用的时候肯定要依靠其子类进行实例化操作。使用“GregorianCalendar”类就可以完成功能。

```
public class CalendarDemo {
    public static void main(String[] args) {
        //两种实例化方式
        Calendar c = Calendar.getInstance();
        //Calendar c = new GregorianCalendar();
        System.out.println("年："+c.get(Calendar.YEAR));
        System.out.println("月："+(c.get(Calendar.MONTH)+1));
        System.out.println("日："+c.get(Calendar.DAY_OF_MONTH));
        System.out.println("小时："+c.get(Calendar.HOUR));
        System.out.println("分钟："+c.get(Calendar.MINUTE));
        System.out.println("秒："+c.get(Calendar.SECOND));
```

```
        System.out.println("毫秒："+c.get(Calendar.MILLISECOND));
        String[] weeks = {"星期日","星期一","星期二","星期三","星期四","星期五","星期六"};
        System.out.println("星期："+weeks[c.get(Calendar.DAY_OF_WEEK)-1]);
    }
}
```

3.4.3、DateFormat

java.text.DateFormat 类表示的是日期的格式化类，可以将一个日期按照指定的风格进行格式化的操作。

```
public abstract class DateFormat extends Format
```

DateFormat 类本身也属于抽象类，但是此抽象类使用的时候可以不经过子类实例化，在此类中定义了如下的操作：

No.	方法名称	类型	描述
1	public static final DateFormat getDateInstance()	普通	得到默认的对象
2	public static final DateFormat getDateInstance(int style,Locale aLocale)	普通	根据 Locale 的指定区域得到对象
3	public static final DateFormat getDateTimeInstance()	普通	得到默认的日期时间对象
4	public static final DateFormat getDateTimeInstance(int dateStyle,int timeStyle,Locale aLocale)	普通	得到指定 Locale 的日期、时间对象
5	public final String format(Date date)	普通	对日期进行格式化的操作

但是在讲解之前需要注意的是 DateFormat 类本身也属于 Format 类的子类。之前讲解 MessageFormat 类的时候曾经说过 MessageFormat 类也是 Format 的子类。

一个国际化的操作不光只有文字，还有货币显示和日期的显示风格。

```
package org.dateformatdemo;
import java.text.DateFormat;
import java.util.Date;
public class DateFormatDemo01 {
    public static void main(String[] args) {
        DateFormat df1 = DateFormat.getDateInstance();// 得到默认的对象
        DateFormat df2 = DateFormat.getDateTimeInstance();// 得到默认的对象
        System.out.println(df1.format(new Date())); // 格式化日期时间
        System.out.println(df2.format(new Date())); // 格式化日期时间
    }
}
```

此时的日期格式，采用的是本机默认的语言环境。当然，也可以为其加入一些其他的地区标记。

```
package org.dateformatdemo;
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;
public class DateFormatDemo02 {
    public static void main(String[] args) {
        DateFormat df1 = DateFormat.getDateInstance(DateFormat.YEAR_FIELD,
                new Locale("en", "US"));// 得到默认的对象
        DateFormat df2 = DateFormat.getDateTimeInstance(DateFormat.YEAR_FIELD,
                DateFormat.ERA_FIELD, new Locale("en", "US"));// 得到默认的对象
        System.out.println(df1.format(new Date())); // 格式化日期时间
        System.out.println(df2.format(new Date())); // 格式化日期时间
    }
}
```

所以，一般来讲在开发中比较常用的操作就是 DateFormat 类的子类：SimpleDateFormat 类。

3.4.4、SimpleDateFormat

SimpleDateFormat 类的主要功能是完成日期显示格式的转换，例如，现在有如下的一个日期时间：

- 原始日期：2009-03-31 14:55:42.718
- 转换日期：2009 年 03 月 31 日 14 时 55 分 42 秒 718 毫秒

那么，这个时候就必须使用 SimpleDateFormat 类完成功能，但是在转换的时候需要采用如下的步骤：

1、 指定一个模板，并根据此模板，从第一个日期中取出所有的时间数字

2、 所有的时间数字将使用 Date 保存

3、 将所有的时间数字重新进行格式的转换

如果要想将全部的日期-时间数组取出，则必须定义相应的操作模板，此模板要求如下：

No.	日期	模板	描述
1	年	y	表示年，年一般是四位数字，所以需要使用“yyyy”表示
2	月	M	表示月，月一般是二位数字，所以需要使用“MM”表示
3	日	d	表示日，日一般是二位数字，所以需要使用“dd”表示
4	时	HH	表示时，时一般是二位数字，所以需要使用“HH”表示
5	分	mm	表示分，分一般是二位数字，所以需要使用“mm”表示
6	秒	ss	表示秒，秒一般是二位数字，所以需要使用“ss”表示

7	豪秒	S	表示毫秒，豪秒一般是三位数字，所以需要使用“SSS”表示

范例：下面进行模板的转换

```
package org.dateformatdemo;
import java.text.SimpleDateFormat;
import java.util.Date;
public class SimpleDateFormatDemo01 {
    public static void main(String[] args) throws Exception {
        String str = "2009-03-31 14:55:42.718";
        String pat1 = "yyyy-MM-dd HH:mm:ss.SSS";
        String pat2 = "yyyy年MM月dd日 HH时mm分ss秒SSS毫秒";
        SimpleDateFormat sdf1 = new SimpleDateFormat(pat1); // 定义第一个模板
        SimpleDateFormat sdf2 = new SimpleDateFormat(pat2); // 定义第一个模板
        Date date = sdf1.parse(str); // 将字符串的日期时间取出
        String newDate = sdf2.format(date); // 重新进行新的格式套用
        System.out.println(newDate);
    }
}
```

3.5、比较器（重点）

3.5.1、问题的引出

java.util.Arrays 类本身可以完成数组的排序功能，在此类中定义了如下的一个方法：

- 为对象数组排序 public static void sort(Object[] a)

例如：现在有如下的操作类

```
package org.compareabledemo01;
public class Person {
    private String name ;
    private int age ;
    public Person(){}
    public Person(String name,int age){
        this.setName(name) ;
        this.setAge(age);
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
```

```
    public void setAge(int age) {
        this.age = age;
    }
    public String toString() {
        return "姓名: " + this.name + "\t" + "年龄: " + this.age;
    }
}
```

之后，要求产生 6 个此类的对象，并定义成对象数组，并进行排序的操作。

```
package org.compareabledemo01;
import static java.util.Arrays.*; // 静态导入
public class ComparableDemo01 {
    public static void main(String[] args) {
        Person[] per = { new Person("张三", 18), new Person("李四", 22),
                new Person("王五", 10), new Person("赵六", 30),
                new Person("孙七", 33), new Person("王八", 90) };
        sort(per); // 排序操作
        fun(per); // 打印输出
    }
    public static void fun(Person p[]) {
        for (Person x : p) {
            System.out.println(x);
        }
    }
}
```

以上程序的操作语法并没有任何的问题，下面执行程序，出现了以下的错误：

```
Exception in thread "main" java.lang.ClassCastException: org.compareabledemo01.Person
cannot be cast to java.lang.Comparable
    at java.util.Arrays.mergeSort(Arrays.java:1144)
    at java.util.Arrays.sort(Arrays.java:1079)
    at org.vince.compareabledemo01.ComparableDemo01.main(ComparableDemo01.java:10)
```

出现了类型转换异常，但是所有的代码中并没有类型转换。

从 Arrays 类的 sort()方法的定义上可以发现，如果没有实现 Comparable 接口会造成此问题。那么为什么要实现此接口呢？

思考：之前进行数字排序，应该按照大小进行排序的操作，但是现在给的是对象。对象现在有排序规则吗？

3.5.2、排序规则 —— Comparable 接口

所有的排序规则必须依靠 Comparable 接口完成，此接口定义格式如下：

```
public interface Comparable<T>{
    public int compareTo(T o) ;
}
```

以上的方法是 compareTo()方法，此方法是作为比较操作存在的，但是此方法的返回值是 int 型数据。此方法有三种返回结果：

- 返回 0：表示相等
- 返回 1：表示大于

- 返回-1：表示小于

所以，此时，如果要想正确的进行排序操作的话，则 Person 类必须实现 Comparable 接口。

```
package org.compareabledemo02;
public class Person implements Comparable<Person> {
    private String name;
    private int age;
    public int compareTo(Person o) {
        if (this.age > o.age) {
            return 1;
        } else if (this.age < o.age) {
            return -1;
        } else {
            return 0;
        }
    }
    public Person() {
    }
    //setter及getter方法
    public String toString() {
        return "姓名: " + this.name + "\t" + "年龄: " + this.age;
    }
}
```

下面再次运行比较程序，观察结果。

只要以后牵扯到对象的比较，则永远都需要 Comparable 接口完成。

3.5.3、深入研究 Comparable 接口（理解）

从 Comparable 操作的 compareTo()方法上可以发现，此方法返回的有三种值：-1、0、1，那么实际上这种排序规则就非常类似于数据结构中学习过的 BT（Binary Tree）算法。

现在假设有如下的一组数据：“5、6、1、2、3、90、35、7”，要求现在使用 BT 算法排序。

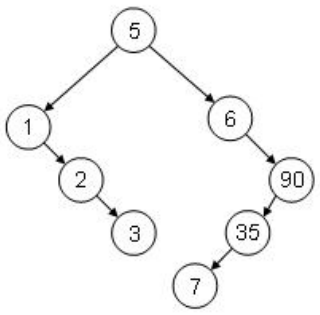
二叉树算法的排序规则：

1、 选择第一个元素作为根节点

2、 之后如果元素大于根节点放在右子树，如果元素小于根节点，则放在左子树

3、 最后按照中序遍历的方式进行输出，则可以得到排序的结果（左→根→右）

5、6、1、2、3、90、35、7



中序遍历之后结果是：1 → 2 → 3 → 5 → 6 → 7 → 35 → 90

此时，排序的功能完成。

在执行之前先来观察以下的一种操作代码：

```
package org.compareabledemo03;
public class ComDemo {
    public static void main(String[] args) {
        Comparable<String> com1 = "hello"; // 向上转型
        Comparable<Integer> com2 = 1; // 向上转型
        System.out.println(com1);
        System.out.println(com2);
    }
}
```

只要实现了 Comparable 接口的对象，都可以使用 Comparable 接口进行内容的接收。

既然明白之后，那么下面动手自己手工实现一个二叉树。

```
package org.compareabledemo03;
class BinaryTree { // 定义二叉树的操作类
    class Node {
        private Comparable data; // 保存数据
        private Node left;// 表示左子树
        private Node right;// 表示右子树
        public Node(Comparable data) {
            this.data = data;
        }
        public void addNode(Node newNode) {
            if (newNode.data.compareTo(this.data) < 0) {
                if (this.left == null) { // 当前的左子树是否等于空
                    this.left = newNode;
                } else {
                    this.left.addNode(newNode);// 继续向下继续判断
                }
            }
            if (newNode.data.compareTo(this.data) >= 0) {
                if (this.right == null) { // 当前的右子树是否等于空
                    this.right = newNode;
                } else {
                    this.right.addNode(newNode);
                }
            }
        }
        public void printNode() {
            if (this.left != null) {
                this.left.printNode();
            }
            System.out.println(this.data);
            if (this.right != null) {
                this.right.printNode();
```

```
            }
        }
    }
    private Node root; // 定义根节点
    public void add(Comparable data) { // 表示增加节点
        Node newNode = new Node(data);
        if (this.root == null) { // 此时没有根节点，第一个元素作为根节点
            this.root = newNode;
        } else { // 判断节点是放在左子树还是右子树
            this.root.addNode(newNode);
        }
    }
    public void print() { // 打印节点
        this.root.printNode();
    }
}
public class BinaryTreeDemo {
    public static void main(String[] args) {
        BinaryTree bt = new BinaryTree();
        bt.add(5);
        bt.add(3);
        bt.add(1);
        bt.add(90);
        bt.add(90);
        bt.add(100);
        bt.add(60);
        bt.print();
    }
}
```

在面试之前最好将此程序巩固一下，因为一个是链表，一个是二叉树是最有可能在笔试中出现的题目。

3.5.4、Comparator（理解）

以上的操作代码中，基本上都是在类定义的时候已经加入了比较器的操作，那么如果现在一个类已经定义完成了，再加入比较器，则肯定会很麻烦。

在 java.util.Arrays 类中存在以下的一个方法：public static <T> void sort(T[] a, Comparator<? super T> c)

此方法用于接收 java.util.Comparator 的比较操作，Comparator 实际上就属于一个挽救的比较器接口。

```
package org.compareabledemo04;
import java.util.Comparator;
public class PersonComparator implements Comparator<Person> {
    public int compare(Person o1, Person o2) {
        if (o1.getAge() > o2.getAge()) {
            return 1;
        } else if (o1.getAge() < o2.getAge()) {
            return -1;
```

```
        } else {
            return 0;
        }
    }
}
```

此时 Person 类已经开发完成了，以上的操作类只是挽救了一个比较的操作而已。

```
package org.vince.compareabledemo04;
import static java.util.Arrays.*; // 静态导入
public class ComparatorDemo {
    public static void main(String[] args) {
        Person per[] = { new Person("张三", 18), new Person("李四", 22),
                new Person("王五", 10), new Person("赵六", 30),
                new Person("孙七", 33), new Person("王八", 90) };
        sort(per,new PersonComparator()); // 排序操作
        fun(per); // 打印输出
    }
    public static void fun(Person[] p) {
        for (Person x : p) {
            System.out.println(x);
        }
    }
}
```

两种操作比较起来，Comparable 接口使用的是最多的，而对于 Comparator 接口一般使用较少，因为要开发两个类比较麻烦。

3.6、对象的克隆技术（理解）

将一个对象复制一份，称为对象的克隆技术。

在 Object 类中存在一个 clone()方法：protected Object clone() throws CloneNotSupportedException

但是，并不是所有的对象都具备克隆的功能，所以，如果某个类的对象要想被克隆，则对象所在的类必须实现 Cloneable 接口。但是此接口并没有方法，那么此接口实际上将作为一个标识接口出现。

```
package org.vince.clonedemo;
public class Person implements Cloneable { // 表示可以被克隆
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
```

```
    public void setAge(int age) {
        this.age = age;
    }
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
    public String toString() {
        return "姓名: " + this.name + ", 年龄: " + this.age;
    }
}
```

下面对程序代码进行测试：

```
package org.clonedemo;
public class ClonePerson {
    public static void main(String[] args) throws CloneNotSupportedException {
        Person per = new Person();
        per.setName("张三");
        per.setAge(20);
        Person p = (Person) per.clone();
        p.setName("李四");
        System.out.println(per);
        System.out.println(p);
    }
}
```

此时，证明对象已经被成功的克隆了。

4、总结

1、 String 与 StringBuffer 的区别

2、 Date 可以取得一个时间，SimpleDateFormat 可以对时间进行格式的转换

4、 Comparable 比较器的使用，了解其基本的操作原理

5、 了解国际化程序的实现

5、作业

1、 完成本节所有示例代码。（提高代码量是成为 IT 民工的最重要标准）。