JavaSE

1. 数据类型

1.1. 基础:int /short/long/double/char/float/byte/boolean

1.2. 包装类:Integer/Short/Long/Double/Char/Float/Byte/Boolean

1.3. 在许多情况下包装与解包装是由编译器自行完成的(当需要往ArrayList，Hash Map中放东西时，基本类型是放不进去的，因为容器都是装object的，这是就需要这些基本类型的包装器类了。)

2. 类

2.1. 抽象类/接口/普通类/

2.1.1. 抽象类:的抽象方法只有定义,没有实现;不能实例化对象;可以没有抽象方法

2.1.2. 接口:JDK1.8以后

接口包含有抽象方法,静态方法,默认方法;;;一些不需要重写的方法我们我们就直接在接口中定义好,通过default关机键字实现接口中的默认方法

2.1.3. 普通类:方法/代码块/

方法重载:就是方法名相同,方法参数的个数和类型不同,通过个数和类型的不同来区分不同的函数;

方法的重载跟返回值类型和修饰符无关,Java的重载是发生在本类中的,重载的条件实在本类中有多个方法名相同,但参数列表不同(可能是,参数个数不同参数类型不同)跟返回值无关

静态代码块:随着类的加载加载,只加载一次

2.2. 面向对象

2.2.1. 封装

JavaBean规范:一般成员变量是private修饰保证数据安全,别的类无法访问这些变量,提供getter.setter取值赋值

2.2.2. 多态

使用多态的好处:方法的扩展性强;利于代码维护;常用Map m= new HashMap();List l =new ArrayList();

2.2.3. 继承

子类继承父类，子类可以继承父类中具有访问控制权限的属性和方法（一般来说是非private修饰的），对于private修饰的父类所特有的属性和方法，子类是不继承过来的。当子类需要改变继承过来的方法时，也就是常说的重写父类的方法。一旦重写后，父类的此方法对子类来说表现为隐藏。以后子类的对象调用此方法时，都是调用子类重写后的方法

3. 数组 /集合

3.1. 数组

3.1.1. Java在定义数组时并不为数组元素分配内存，因此[ ]中无需指定数组元素的个数

数组长度静态数组容量固定的缺点，实际开发中使用频率不高，被 ArrayList 或 Vector 代替

3.2. 集合

3.2.1. 1) Collection 每个位置只能保存一个元素(对象)

2)

Map保存的是"键值对"，就像一个小型数据库。我们可以通过"键"找到该键对应的"值"

3.2.2. Collection

List集合代表一个元素有序、可重复的集合，集合中每个元素都有其对应的顺序索引。List集合允许加入重复元素，因为它可以通过索引来访问指定位置的集合元素。List集合默认按元素
的添加顺序设置元素的索引

Set集合类似于一个罐子，"丢进"Set集合里的多个对象之间没有明显的顺序。Set继承自Collection接口，不能包含有重复元素(记住，这是整个Set类层次的共有属性)。Set判断两个对象相同不是使用"=="运算符，而是根据equals方法。也就是说，我们在加入一个新元素的时候，如果这个新元素对象和Set中已有对象进行注意equals比较都返回false，
则Set就会接受这个新元素对象，否则拒绝。

3.2.3. Map

Map用于保存具有"映射关系"的数据，因此Map集合里保存着两组值，一组值用于保存Map里的key，另外一组值用于保存Map里的value。key和value都可以是任何引用类型的数据。Map的key不允许重复，即同一个Map对象的任何两个key通过equals方法比较结果总是返回false。
Map的这些实现类和子接口中key集的存储形式和Set集合完全相同(即key不能重复)

4. IO

4.1. IO流常用基类

4.1.1. 字节流

输出字节流 : OutputStream：字节输出流抽象基类

FileOutputStream：字节输出流

BufferedOutputStream：字节输出流缓冲区

PrintStream : 打印流

输入字节流：InputStream：字节读取流抽象基类

FileInputStream : 字节读取流

BufferedInputStream : 字节读取流缓冲区

4.1.2. 字符流

输出字符流 : Writer : 字符输出流抽象基类

FileWriter : 字符输出流

BufferedWriter：字符输出流缓冲区

OutputStreamWriter : 字符通向字节的转换流(涉及键盘录入时用)

PrintWriter : 打印流,可处理各种类型的数据

输入字符流 : Reader: 字符读取流的抽象基类

FileReader：字符读取流

BufferedReader：字符读取流缓冲区

InputStreamReader：字节通向字符的转换流(涉及键盘录入时用)

4.2. IO流常见需求

4.2.1. 字符流

需求1 : 在硬盘上创建一个文件并写入信息

用字符写入流：FileWriter

```
FileWriter fw = new FileWriter("g:\\filewriter.txt");
fw.write("输入信息");
fw.write("也可以写入字符数组".toCharArray());
fw.flush();
fw.close();
```

需求2 : 在原有文件上续写数据

```
FileWriter fw = new FileWriter("g:\\filewriter.txt",true);
fw.write("还可以续写信息");
fw.write("也可以写入字符数组".toCharArray());
fw.flush();
fw.close();
```

需求3 : 读取硬盘上的文本文件,并将数据打印在控制台

```
FileReader fr = new FileReader("g:\\filewriter.txt");
**第一种读取方法：一个一个字节的读
int ch = 0;
ch = fr.read();
while((ch = fr.read()) != -1){
   sop((char)ch);
}
fr.close();
**第二种读取方法：利用数组来提高效率
char[] buf = new char[1024];
int len = 0;
while((len = fr.read(buf))!=-1)
{
```

```
    sop(new String(buf,0,len));
  }
  fr.close();
```

需求4 : 拷贝文本文件

利用缓冲区提高数据读写效率(无缓冲区就相当于一滴一滴的喝水，有缓冲区就相当于一杯一杯的喝水)

```
BufferedReader bufr = new BufferedReader(new
FileReader("g:\\filewriter.txt"));
BufferedWriter bufw = new BufferedWriter(new
FileWriter("d:\copyfilewriter.txt"));
String line = null;
while((line = bufr.readLine())!=null)
{
 burw.write(line);
 bufw.newLine();
 bufw.flush();
}
 bufr.close();
 bufw.close();
```

4.2.2. 字节流

需求1：在硬盘上创建一个文件并写入信息(字节流写入时没有刷新)

```
FileOutputStream fos = new FileOutputStream("g:\\filestream.txt");
fos.write(97);//写入一个字节,int：97代表写入char：a
fos.write("也可以写入字节数组".getBytes());//通常使用此种方式写入，直观!
fos.close();
```

需求2：在硬盘已有文件上续写数据(字节流写入时没有刷新)

```
FileOutputStream fos = new FileOutputStream("g:\\filestream.txt",true);
fos.write("创建字节写入流时，传进去一个true参数就可以继续写入信息".g
etBytes());
fos.close();
```

需求3：读取硬盘上的文件

```
FileInputStream fis = new FileInputStream("g:\\filestream.txt");
**第一种读法：一个字节一个字节的读(此种读法慢)
int ch = 0;
while((ch = fis.read())!=-1)
{
  sop((char)ch);
}
fis.close();
**第二种读法：利用字节数组读(此种读法效率有一定提高)
byte[] buf = new byte[1024];
int len = 0;
while((len = fis.read())!=-1)
{
  sop(new String(buf,0,len));
}
fis.close();
```

需求4:拷贝字节文件，如图片或者MP3或者电影

```
**第一种拷贝：带缓冲区，高效
FileInputStream fis = new FileInputStream("g:\\1.mp3");
FileOutputStream fos = new FileOutputStream("g:\\copy1.mp3");
byte[] buf = new byte[1024];
```

```
int len = 0;
while((len = fis.read(buf))!=-1)
{
  fos.(buf,0,len);//字节流写入无需刷新
}
fis.close();
fos.close();
**第二种拷贝：不带缓冲区(慢，还是效率问题)
BufferedInputStream bufi = new BufferedInputStream(new
FileInputStream("g:\\1.mp3"));
BufferedOutputStream bufo = new
BufferedOutputStream(newFileOutputStream("g:\\copy1.mp3"));
int ch = 0;
while((ch = bufi.read())!=-1)
{
  bufo.write(ch);
}
bufi.close();
bufo.close();
```

4.2.3. 转换流

需求1：读取一个键盘录入

```
InputStream in = System.in;//创建一个键盘录入流，流不关则可以一直录入
int by1 = in.read();//一次读一个字节
int by2 = in.read();//一次读一个字节
sop(by1);//假设键盘录入的是abcd,则打印a
sop(by2);//假设键盘录入的是abcd,则打印b
in.close();
```

需求2：键盘录入一行数据打印一行数据，如果录入的是over则结束录入

```
InputStream in = System.in;
StringBuilder sb = new StringBuilder();
    while(true)
    {
      int ch = in.read();
      if(ch=='\r')
        continue;
      if(ch=='\n')
      {
        String line = sb.toString();
        if("over".equals(line))
          break;
        sop(line.toUpperCase());//输出大写
        sb.delete(0.sb.length());//清除上一行录入的数据

      }
      else
        sb.append((char)ch);
    }
    in.close();
```

需求3：发现需求2中其实就是读一行的原理，故引入字节通向字符的桥梁：InputStreamReader为提高效率加入缓冲区

```
BufferedReader bufr = new BufferedReader(new
InputStreamReader(System.in));
String line = null;
    while((line = bufr.readLine())!=null)
    {
      if("over".equals(line))
```

```
            break;
          sop(line.toUpperCase());//输出大写
        }
        bufr.close();
```

需求4：键盘录入数据并打印到控制台

```
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
BufferedWriter bw=new BufferedWriter(new
OntputStreamWriter(System.out));
      String line = null;
      while((line = br.readLine())!=null)
      {
        if("over".equals(line))
          break;
        bw.write(line.toUpperCase());
        bw.newLine();
        bw.flush();
      }
      br.close();
      bw.close();
```

需求5:将键盘录入的数据存储到硬盘文件

则只需将(4)中的BufferedWriter bufw = new BufferedWriter(new OntputStreamWriter(System.out));改为：BufferedWriter bufw = new BufferedWriter(new OntputStreamWriter(new FileWriter("g:\\demo.txt")));

即:

```
BufferedReader bufr = new BufferedReader(new
InputStreamReader(System.in));
BufferedWriter bufw = new BufferedWriter(new OntputStreamWriter(new
FileWriter("g:\\demo.txt")));
```

```
String line = null;
while((line = bufr.readLine())!=null)
{
  if("over".equals(line))
    break;
  bufw.write(line.toUpperCase());
  bufw.newLine();
  bufw.flush();
}
bufr.close();
bufw.close();
```

需求6：将硬盘文件的数据打印到控制台

则只需将(4)中的

```
BufferedReader bufr = new BufferedReader(new
InputStreamReader(System.in));
```

改为：

```
BufferedReader bufr = new BufferedReader(new InputStreamReader(new
FileReader("g:\\demo.txt")));
```

即：

```
BufferedReader bufr = new BufferedReader(new InputStreamReader(new
FileReader("g:\\demo.txt")));
BufferedWriter bufw = new BufferedWriter(new
OntputStreamWriter(System.out));
    String line = null;
    while((line = bufr.readLine())!=null)
    {
      if("over".equals(line))
        break;
      bufw.write(line.toUpperCase());
```

```
            bufw.newLine();
            bufw.flush();
        }
        bufr.close();
        bufw.close();
```

4.3. File类

4.3.1. 构造方法

File(String pathname)

通过将给定路径名字符串转换为抽象路径名来创建一个新 **File** 实例

File(String parent, String child) 根据 **parent** 路径名字符串和 **child** 路径名字符串创建一个新 **File** 实例

File(File parent, String child) 根据 **parent** 抽象路径名和 **child** 路径名字符串创建一个新 **File** 实例

4.3.2. 方法摘要

(1)创建：

boolean createNewFile()

当且仅当不存在具有此抽象路径名指定名称的文件时，不可分地创建一个新的空文件

boolean mkdir() 创建一级文件夹

boolean mkdirs() 创建多级文件夹

(2)判断：

boolean isDirectory() 测试此抽象路径名表示的文件是否是一个目录

boolean isFile() 测试此抽象路径名表示的文件是否是一个标准文件

boolean exists() 测试此抽象路径名表示的文件或目录是否存在

(3)获取：

String getParent()

返回此抽象路径名父目录的路径名字符串；如果此路径名没有指定父目录，则返回 **null**

File getParentFile()

返回此抽象路径名父目录的抽象路径名；如果此路径名没有指定父目录，则返回 **null**

String getName() 返回由此抽象路径名表示的文件或目录的名称

String getPath() 将此抽象路径名转换为一个路径名字符串

(4)删除：

boolean delete() 删除此抽象路径名表示的文件或目录

(5)获取全部：**(**非常重要！！！**)**

String[] list()

返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中的文件和目录

File[] listFiles()

返回一个抽象路径名数组，这些路径名表示此抽象路径名表示的目录中的文件

String[] list(FilenameFilter filter)

返回一个字符串数组，这些字符串指定此抽象路径名表示的目录中满足指定过滤器的文件和目录

File[] listFiles(FileFilter filter)

返回抽象路径名数组，这些路径名表示此抽象路径名表示的目录中满足指定过滤器的文件和目录

5. 异常

5.1.1、try块中的局部变量和catch块中的局部变量（包括异常变量），以及finally中的局部变量，他们之间不可共享使用。

5.2.2、每一个catch块用于处理一个异常。异常匹配是按照catch块的顺序从上往下寻找的，只有第一个匹配的catch会得到执行。匹配时，不仅运行精确匹配，也支持父类匹配，因此，如果同一个try块下的多个catch异常类型有父子关系，应该将子类异常放在前面，父类异常放在后面，这样保证每个catch块都有存在的意义。

5.3.3、java中，异常处理的任务就是将执行控制流从异常发生的地方转移到能够处理这种异常的地方去。也就是说：当一个函数的某条语句发生异常时，这条语句的后面的语句不会再执行，它失去了焦点。执行流跳转到最近的匹配的异常处理catch代码块去执行，异常被处理完后，执行流会接着在“处理了这个异常的catch代码块”后面接着执行。