计算三角形三个角的角度

计算三角形的三个角的角度

- 输入二维空间三个点的坐标
  - (xi, yi), i=1,2,3
- 计算三个角的角度
  - 从顶点座标计算三条边的长度
  - 由3条边的长度计算夹角余弦
  - 由反余弦函数求夹角的角度
- java.lang.Math类

其它
double sqrt(double d) //返回平方根
double pow(double d1, double d2)
double random()

三角等
请参考相关帮助文档

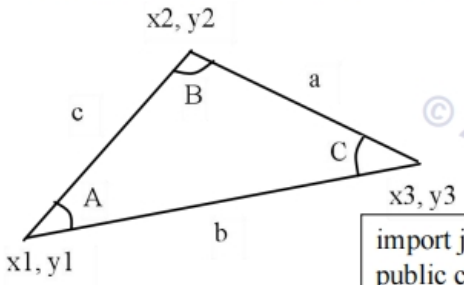
截取
double ceil(double d) //返回不小于d的最小整数
double floor(double d) //返回不大于d的最大整数
int round(float f) //返回四舍五入后的整数
long round(double d) //返回四舍五入后的整数

常数
PI //double，圆周率 E //double，自然对数

变换（int, long, float各种类型相似）
double abs(double d) //返回绝对值
double min(double d1, double d2)
double max(double d1, double d2)

- Java中的数学（Math）类是final类，不可继承。
- 其中包含一组静态方法和两个常数

Java中的异常




```
A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))
```

程序操作的数据必须位于计算机内存，并且有特定的类型的存储格式，否则程序无法正常操作数据。I/O就是对输入和输出程序的数据进行操作和抽象

Math.round(A * 100) / 100.0
--保留两位小数

```
import java.util.Scanner;
public class ComputeAngles {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter three points: ");
    double x1 = input.nextDouble();   double y1 = input.nextDouble();
    double x2 = input.nextDouble();   double y2 = input.nextDouble();
    double x3 = input.nextDouble();   double y3 = input.nextDouble();
    double a = Math.sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3));
    double b = Math.sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
    double c = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
    double A = Math.toDegrees(Math.acos((a * a - b * b - c * c) / (-2 * b * c)));
    double B = Math.toDegrees(Math.acos((b * b - a * a - c * c) / (-2 * a * c)));
    double C = Math.toDegrees(Math.acos((c * c - b * b - a * a) / (-2 * a * b)));
    System.out.println("The three angles are " + Math.round(A * 100) / 100.0 + " " +
        Math.round(B * 100) / 100.0 + " " + Math.round(C * 100) / 100.0);
  }
}
```

回顾

命令行参数

- 复习Java命令行带参数
  - Java程序启动时，可以添加0或多个命令行参数（Command-ling Arguments）
  - 不管使用双引号与否都作为字符串自动保存到main函数的参数中

```
public class TestArgs{
  public static void main(String args[]){
    System.out.println(args.length);
    for(int i=0; i<args.length; i++){
      System.out.println(args[i]);
    }
  }
}
```

```
java TestArgs arg1  arg2  "another arg"
java TestArgs arg1+arg2
```

也有在Eclipse环境中的run configurations…中指定参数

- 命令行参数可以向程序输入信息
  - 但必须在程序运行前预先设定好，且运行程序的人必须理解程序的参数要求
  - 一般只用于参数易于理解，或程序员调试程序

在程序运行过程中，用户根据提示输入信息，增加了程序的灵活性

流的概念

- 流就是字节序列的抽象概念
  - 输入流：能*被连续读取数据*的数据源
  - 输出流：能*被连续写入数据*的接收端
- I/O流有多种，按操作数据单位不同可分为**字节流(8b)和字符流(16b)**
- 流机制是Java及C++的重要机制，通过流开发人员科研自由控制文件、内存、I/O设备等数据流向
  - I/O流就是用于处理设备上的数据，如硬盘、内存、键盘录入等，如同管道，将两个容器连接起来

Java I/O流涉及40多个类，实际上这些类都是由这4个抽象基类派生

	字节流	字符流
输入流	inputStream	Reader
输出流	outputStream	Writer

字节流

- `InputStream`和`OutputStream`都是抽象类，不能实例化
  - 按字节处理数据

JAVA

```java
package lesson11;
import java.io.*;

public class FileCopyWithStream {
        public static void main(String[] args) throws IOException{
                File srcFile = new File("src/lesson11/FileCopyWithStream.java");
                FileInputStream fis = new FileInputStream(srcFile);

                File destFile = new File("backup.java");
                // 没有指定路径，这个文件被放在src的上级目录
                FileOutputStream fos = new FileOutputStream(destFile);

                byte b[] = new byte[512];  // 创建缓冲区
                int len;
                long begin = System.currentTimeMillis();
                while((len = fis.read(b)) != -1) {  //当读取长度为-1时，表示文件结束
                        fos.write(b, 0, len);
                        //每次读取时，使用fos.write()将读取的字节写入目标文件
                }
                long end = System.currentTimeMillis();
                System.out.println("copy file spent time: " + (end - begin) + "
milliseconds");
                fis.close();
                fos.close();
        }
}
```

这份代码的作用是复制`FileCopyWithStream`的全部内容，并将其拷贝到`backup.java`的文件中

1. `File srcFile = new File("src/lesson11/FileCopyWithStream.java");`：创建了一个`File`对象，代表**源文件**。这里的源文件是`src/lesson11/FileCopyWithStream.java`。
2. `FileInputStream fis = new FileInputStream(srcFile);`：创建了一个`FileInputStream`对象，用于从源文件中读取数据。

Q：为什么需要创建一个`FileInputStream`对象，直接读取不行吗？

创建`FileInputStream`对象是为了**与文件进行交互**，它提供了读取文件内容的方法。在Java中，**不能直接读取文件内容**，需要通过**输入流**来实现。`FileInputStream`是Java I/O库中的一个类，专门用于从文件中读取数据。当你创建一个`FileInputStream`对象并关联到一个文件时，你就打开了一个指向该文件的输入流，然后可以通过这个输入流来读取文件中的数据。

3. `File destFile = new File("backup.java");`：创建了一个`File`对象，代表**目标文件**。这里的文件名是`backup.java`，并且因为它没有指定路径，所以它将被创建在源文件的同级目录下。

4. `FileOutputStream fos = new FileOutputStream(destFile);`：创建了一个`FileOutputStream`对象，用于将数据写入目标文件。

5. `byte b[] = new byte[512];`：创建了一个字节数组，作为**缓冲区**。这个缓冲区的大小是512字节。

Q：什么是缓冲区？为什么要使用`byte`数组作为缓冲区？数组长度对缓冲区有什么影响？

在文件复制过程中，使用byte数组作为缓冲区可以提高效率。缓冲区是一个存储数据的临时区域，它位于内存中，

用于在读取和写入操作之间暂存数据。

使用byte数组作为缓冲区的好处有以下几点：

1. **减少I/O次数**：如果每次只读取或写入一个字节，那么对于大文件来说，将需要非常多次的I/O操作。通过使用缓冲区，我们可以*一次性读取或写入多个字节*，从而大大减少I/O次数。
2. **提高效率**：由于*减少了I/O次数*，所以整体的文件复制速度会得到提高。
3. **灵活性**：使用缓冲区*可以更容易地处理各种数据块的大小*。

数组的长度确实会对缓冲区有影响：

- **如果数组长度太小**：例如，如果数组长度只有1字节，那么缓冲区的优势就几乎不存在了，因为每次只能处理一个字节。这样会增加I/O操作的次数，从而降低效率。
- **如果数组长度适中**：例如，常见的选择是512字节或1024字节。这样的缓冲区大小可以在大多数情况下提供很好的性能。
- **如果数组长度太大**：虽然大缓冲区可以减少I/O次数，但它也会占用更多的内存。如果缓冲区过大，可能会导致内存不足的问题，尤其是在处理多个大文件或同时运行其他内存密集型应用时。

3. `int len;`：声明了一个整数变量，用于存储从源文件中*读取的字节数*。
4. `long begin = System.currentTimeMillis();`：获取当前时间的毫秒数，并存储在变量`begin`中。这用于计算文件复制所需的时间。
5. `while((len = fis.read(b)) != -1) { ... }`：这是一个循环，用于从源文件中读取数据，直到文件的末尾。每次循环，它都尝试从源文件中读取最多512个字节的数据到缓冲区`b`中，并将实际读取的字节数存储在`len`中。当读取到文件的末尾时，`read()`方法返回-1，循环结束。
6. `fos.write(b, 0, len);`：这行代码在循环内，将从源文件中读取的数据（存储在缓冲区`b`中）写入目标文件。它仅写入实际读取的字节数（即`len`个字节）。
7. `long end = System.currentTimeMillis();`：在文件复制完成后，再次获取当前时间的毫秒数，并存储在变量`end`中。
8. `System.out.println("copy file spent time: " + (end - begin) + " milliseconds");`：计算并打印文件复制所需的时间（以毫秒为单位）。

过滤流

过滤流

- InputStream类声明的read()方法按照流中字节的先后顺序读取字节
  - FileInputStream和ByteArrayInputStream等输入流都按照这种方式读数据
- 假如希望进一步扩展读数据的功能
  - 创建FileInputStream等输入流的子类，但是这会大大增加输入流类的数目，使输入流的层次结构更加复杂
  - 创建输入流的装饰器，它本身继承InputStream类，还可以装饰其它输入流
- FilterInputStream以及它的子类的构造方法都有一个InputStream类型的参数，该参数指定需要被装饰的输入流
  - 当用过滤输入流来装饰其他输入流时，最后只需关闭过滤输入流，它的close()方法会调用被装饰的输入流的close()方法
  - FilterInputStream有一个子类DataInputStream，DataInputStream实现了DataInput接口，用于读取基本类型数据，如int、float、long、double和boolean等
  - DataInputStream与DataOuputStream搭配使用，可以按照与平台无关的方式从流中读取基本类型（int、char和long等）的数据
  - 输入输出流可以顺序连接（通过前一个流对象放到后一个流的构造器参数）

```java
File fo = new File(strPath);
FileInputStream fos = new FileInputSstream(fo);
BufferOutputStream bos = new BufferOutputStream(fos);
DataOutputStream dos = new DataOutputStream(bos);
```

IO流之间的转换

IO流间的转换

- 字节流中的过滤过滤流FilterInputStream和FilterOutputStream为基础流提供一些额外的功能，常见子类包含
  - 缓冲流(BufferedInputStream/ BufferedOutputStream)：输入流一个字节一个字节地读取，频繁与磁盘进行交互，造成读取速度比较低，缓冲流的存在就是先将数据读取到缓冲流（内存中），然后一次性从内存中读取多个字符，提高读取的效率
  - 数据流(DataInputStream /DataOutputStream)：以机器无关的方式读取Java的基本类型
  - 推回输入流(PushbackInputStream)：Java中读取数据的方式是顺序读取，在流的内部都会维护一个指针，在读取数据的同时，指针会向后移动，直到读取完成为止；而当我们需要将已经读取出来的数据推回输入流则需要引入推回输入流的概念，添加了缓冲区，在需要推回数据时则将数据推回缓冲区，可以重复读入数据

```java
File file = new File(strPath);
FileInputStream fis = new FileInputStream(file);
BufferedInputStream bis = new BufferedInputStream(fis);
```

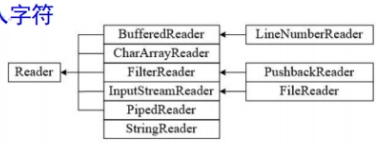
DataInputStream类用法

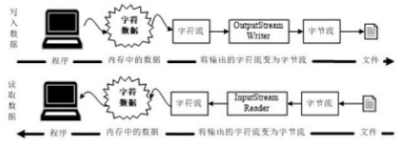
```java
import java.io.*;
public class FormatDataIO{
        public static void main(String[] args)throws IOException{
                FileOutputStream out1 = new FileOutputStream("D:\\test.txt");
                BufferedOutputStream out2 = new BufferedOutputStream(out1); // 装饰文件输出流
                DataOutputStream out = new DataOutputStreaam(out2);
                out.writeByte(-12);
                out.writeLong(12);
                out.writeChar('1');
                out.writeUTF("好");
                out.close();

                InputStream in1 = new FIleInputStream("D:\\test.txt");
                BufferedInputStream in2 = new BufferedInputStream(in1); // 装饰文件输入流
                DataInputStream in = new DataInputStream(in2); // 装饰缓冲输入流
                System.out.print(in.readByte()+" ");
                System.out.print(in.readLong()+" ");
                System.out.print(in.readChar()+" ");
                System.out.print(in.readUTF()+" ");
                in.close();
        }
}
```

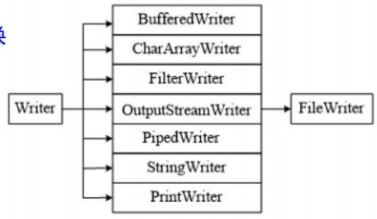
字符流

字符流

- Java还提供了字符流，用于操作字符
  - 与字节流相似，字符流也有两个抽象基类，分别是Reader和Writer
  - Reader是字符输入流，用于从目标文件读取字符
  - Writer是字符输出流，用于向目标文件写入字符
- 字节流和字符流，有时字节流和字符流之间也需要进行转换，在JDK中提供了可以将字节流转换为字符流的两个类
  - InputStreamReader类，字节输入流转换成字符输入流
  - OutputStreamWriter类，字符输出流转换成字节输出流







Reader/Writer

Reader/Writer

- InputStreamReader类把InputStream类型转换为Reader类型
  - **InputStreamReader(InputStream in)**
  - **InputStreamReader reader=new InputStreamReader(new FileInputStream(strPath));**
- BufferedReader(Reader in)：参数in指定被装饰的Reader类
  - **BufferedReader reader = new BufferedReader(new InputStreamReader(...))**
  - 有一个readLine()方法，而BufferedWriter没有相应的writerLine()方法。
- OutputStreamWriter类把OutputStream类型转换为Writer类型
  - **OutputStreamWriter (OutputStream out)**
- BufferedWriter(Writer out)：参数out指定被装饰的Writer类
- PrintWriter能输出格式化的数据，PrintWriter的写数据的方法都以print开头
  - **print(int i); print(long l); print(float f); print(String s)...**
  - **println(int i); println(long l); println(float f); println(String s)...**
- 如果要输出一行字符串，应该用PrintWriter来装饰BufferedWriter，PrintWriter的println(String s)方法可以输出一行字符串

OutputStream out=new FileOutputStream(to); OutputStreamWriter writer=new OutputStreamWriter(out); BufferedWriter bw=new BufferedWriter(writer); PrintWriter pw=new PrintWriter(bw,true);

控制台I/O

打印命令重载

- `System.out`对象默认是计算机屏幕
  - `println`或`print`方法都对8种基本类型进行重载
  - 重载了`char[]`和`Object`

```java
import java.io.*;
public class TestOutput {
        public static void main(String[] args) {
                char c[] = {'a', 'b', 'c'};
                System.out.println(c); // abc
                int d[] = {1,2,3};
                System.out.println(d); // 打印地址
        }
}
```

在键盘实现屏幕输出

```java
import java.io.*;

public class TestInput {
        public static void main(String[] args) {
                InputStreamReader ir = new InputStreamReader(System.in);
                BufferedReader in = new BufferedReader(ir);
                System.out.println("Ctrl + z to exit");
                try {
                        String s = in.readLine();
                        while(s != null) {
                                System.out.println("Read: " + s);
                                s = in. readLine();
                        }
                        in.close();
                }catch(IOException e) {
                        e.printStackTrace();
                }
        }
}
```

文件类

文件类

java.io.File	
+File(pathname: String)	Creates a File object for the specified path name. The path name may be a directory or a file.
+File(parent: String, child: String)	Creates a File object for the child under the directory parent. The child may be a file name or a subdirectory.
+File(parent: File, child: String)	Creates a File object for the child under the directory parent. The parent is a File object. In the preceding constructor, the parent is a string.
+exists(): boolean	Returns true if the file or the directory represented by the File object exists.
+canRead(): boolean	Returns true if the file represented by the File object exists and can be read.
+canWrite(): boolean	Returns true if the file represented by the File object exists and can be written.
+isDirectory(): boolean	Returns true if the File object represents a directory.
+isFile(): boolean	Returns true if the File object represents a file.
+isAbsolute(): boolean	Returns true if the File object is created using an absolute path name.
+isHidden(): boolean	Returns true if the file represented in the File object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character.
+getAbsolutePath(): String	Returns the complete absolute file or directory name represented by the File object.
+getCanonicalPath(): String	Returns the same as getAbsolutePath() except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
+getName(): String	Returns the last name of the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getName() returns test.dat.
+getPath(): String	Returns the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getPath() returns c:\book\test.dat.
+getParent(): String	Returns the complete parent directory of the current directory or the file represented by the File object. For example, new File("c:\\book\\test.dat").getParent() returns c:\book.
+lastModified(): long	Returns the time that the file was last modified.
+length(): long	Returns the size of the file, or 0 if it does not exist or if it is a directory.
+listFile(): File[]	Returns the files under the directory for a directory File object.
+delete(): boolean	Deletes the file or directory represented by this File object.The method returns true if the deletion succeeds.
+renameTo(dest: File): boolean	Renames the file or directory represented by this File object to the specified name represented in dest. The method returns true if the operation succeeds.
+mkdir(): boolean	Creates a directory represented in this File object. Returns true if the the directory is created successfully.
+mkdirs(): boolean	Same as mkdir() except that it creates directory along with its parent directories if the parent directories do not exist.

- File类提供一个抽象描述文件
  - 文件类包含了路径和文件名
  - 文件类有上级文件和下级文件，文件夹也是文件
  - 文件类本身不提供文件读写方法，只表示一个文件对象存在于系统
  - 操作文件内容，需要文件I/O

File

管理文件系统

- File类提供了若干处理文件、目录和获取它们基本信息的方法
- File 类的构造方法有三个：
  - File(String pathname)
  - File(String parent, String child)
  - File(File parent, String child)
- File类的用法
  - 察看文件属性：
    - canRead()
    - isFile()
    - lastModified()
  - 创建或删除目录：mkdir(),delete()
  - 列出目录下的所有文件：list()
  - 判断文件是否存在：exists()
  - 重新命名文件：renameTo()


```
import java.io.*;
impirt java.io.Date;
public class DirUtil{
  public static void main(String args[])throws Exception{
    File dir1=new File("D:\\dir1");
    if(!dir1.exists())dir1.mkdir();

    File dir2=new File(dir1,"dir2");
    if(!dir2.exists())dir2.mkdirs();

    File dir4=new File(dir1,"dir3\\dir4");
    if(!dir4.exists()) dir4.mkdirs();

    File file=new File(dir2,"test.txt");
    if(!file.exists())file.createNewFile();

    listDir(dir1);
    deleteDir(dir1);
  }
```

```java
import java.io.*;
impirt java.io.Date;
public class DirUtil{
        public static void main(String args[])throws Exception{
                File dir1=new File("D:\\dir1");
                if(!dir1.exists())dir1.mkdir();
                File dir2=new File(dir1,"dir2");
                if(!dir2.exists())dir2.mkdirs();
                File dir4=new File(dir1,"dir3\\dir4");
                if(!dir4.exists()) dir4.mkdirs();
                File file=new File(dir2,"test.txt");
                if(!file.exists())file.createNewFile();
                listDir(dir1);
                deleteDir(dir1);
        }
}
```

PrintWriter

java.io.PrintWriter	
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a string.
+print(c: char): void	Writes a character.
+print(cArray: char[]): void	Writes an array of character.
+print(i: int): void	Writes an int value.
+print(l: long): void	Writes a long value.
+print(f: float): void	Writes a float value.
+print(d: double): void	Writes a double value.
+print(b: boolean): void	Writes a boolean value.
Also contains the overloaded println methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded printf methods.	The printf method was introduced in §3.6, "Formatting Console Output and Strings."

Scanner

Scanner按字符输入数据

java.util.Scanner	
+Scanner(source: File)	Creates a Scanner object to read data from the specified file.
+Scanner(source: String)	Creates a Scanner object to read data from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has another token in its input.
+next(): String	Returns next token as a string.
+nextByte(): byte	Returns next token as a byte.
+nextShort(): short	Returns next token as a short.
+nextInt(): int	Returns next token as an int.
+nextLong(): long	Returns next token as a long.
+nextFloat(): float	Returns next token as a float.
+nextDouble(): double	Returns next token as a double.
+useDelimiter(pattern: String): Scanner	Sets this scanner's delimiting pattern.

文件读入屏幕输出

- 文件输入:
  - `FileReader`读字符;
  - `BufferedReader`: `readLine()`方法
- 文件输出:
  - `FileWriter`写字符;
  - `PrintWriter`: `print()`和`println()`方法

JAVA

```java
import java.io.*;
public class ReadFile{
        public static void main(String args[]){
        File file = new File(args[0]);
        try{
                BufferedReader in=new BufferedReader(new FileReader(file));
                String s;
                while((s = in.readLine())!=null){
                System.out.println("Read:"+s);
                }
                in.close();
        }
        catch(FileNotFoundException e1){System.err.println("File not found" + file);}
        catch(IOException e2){e2.printStackTrace();}
        }
}
```

写入文件

JAVA

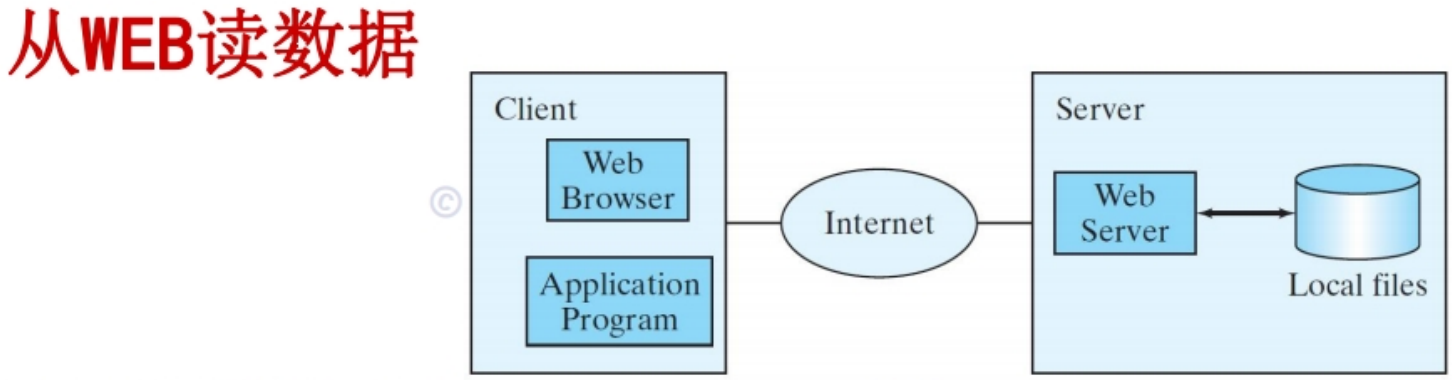
```java
import java.io.*;

public class TestInput {
        public static void main(String[] args) {
                File file = new File("testinput.txt");

                try {
                        InputStreamReader ir = new InputStreamReader(System.in);
                        BufferedReader in = new BufferedReader(ir);
                        PrintWriter out = new PrintWriter(new FileWriter(file));
                        String s;
                        System.out.println("Enter file text and ctrl + z to exit");
                        while((s=in.readLine()) != null) {
                                out.println(s);
                        }
                        in.close();
                        out.close();
                }catch(IOException e) {
                        e.printStackTrace();
                }
        }
}
```

键盘输入文件输出

```java
import java.io.*;
public class WriteFile{
        public static void main(String args[]){
                File file = new File(args[0]);
                try{
                        BufferedReader in=new BufferedReader(new
                        InputStreamReader(System.in));
                        PrintWriter out = new PrintWriter( new FileWriter(file));
                        String s;
                        System.out.println("Enter file text and ctrl+z to exit");
                        while((s = in.readLine())!=null){out.println(s);}
                        in.close(); out.close();
                }catch(IOException e){
                        e.printStackTrace();
                }
        }
}
```

从WEB读入数据

`URL`类



- 从网上的文件读数据
  - 只不过数据不在本地硬盘上，而是在网上某台电脑
- 用URL（Uniform Resource Locator，统一资源定位器）描述文件
  - URL url = new URL("www.google.com/index.html");
- URL对象一旦创新，就可以用URL对象定义的方法 openStream()去读或输入字节流，并用这个流去创建Scanner对象:
  - Scanner input = new Scanner(url.openStream());

文本替换

`ReplaceText`将一个字符串用新字符串替换。字符串和文件名通过命令行参数传递。

JAVA

```
import java.io.*; import java.util.*;
public class ReplaceText {
        public static void main(String[] args) throws Exception {
        if (args.length != 4) {
                System.out.println("Usage: java ReplaceText sourceFile targetFile
oldStr newStr");
                System.exit(1);
        }
        File sourceFile = new File(args[0]);
        if (!sourceFile.exists()) {
                System.out.println("Source file " + args[0] + " does not exist");
System.exit(2);
        }
        File targetFile = new File(args[1]);
        if (targetFile.exists()) {
                System.out.println("Target file " + args[1] + " already exists and
replaced");}
        else {
                System.out.println("A new file is created with replaced text!");
        }
        String str1 = args[2]; String str2 = args[3];
        try(Scanner input = new Scanner(sourceFile); PrintWriter output = new
PrintWriter(targetFile);)
        {
                        while (input.hasNext()) {
                        String s1 = input.nextLine();
                        String s2 = s1.replaceAll(str1, str2);
                        output.println(s2);
                        }
                }
        }
}
```

对象序列化

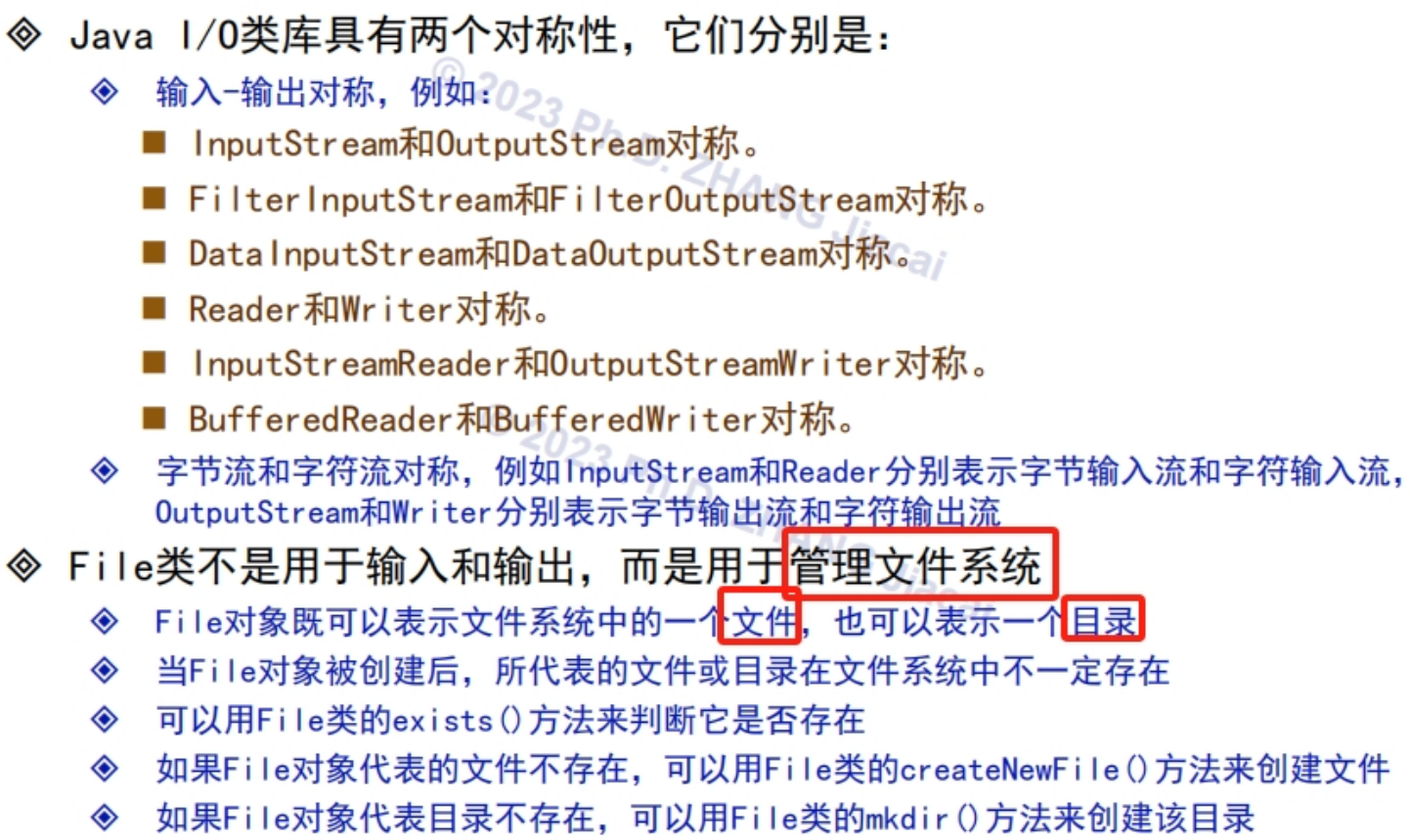
- Java程序运行过程中创建了一系列对象，而且对象的状态也进行了设置，如果程序结束，则所有对象都将被垃圾回收而销毁
- 对象序列化提供了一种技术将对象与硬盘文件互相转化
  - 把**对象转换为字节序列**的过程称为**对象的序列化**（对象打散为字节）
  - 把字节序列恢复为对象的过程称为对象的反序列化（字节结构化为对象）
- 两种用途：
  - 把对象的字节序列**永久地保存**到硬盘上，通常存放在一个文件中
  - 在**网络上传送对**象的字节序列
  - 在很多应用中，需要对某些对象进行序列化，让它们离开内存空间，入住物理硬盘，以便长期保存

JDK类库中的序列化API

- `java.io.ObjectOutputStream`代表对象输出流

    - 其方法`writeObject(Object obj)`方法可对参数指定的obj对象进行序列化，把得到的字节序列写到一个目标输出流中
- `java.io.ObjectInputStream`代表对象输入流
    - 其方法`readObject()`方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回
- 方法不需要序列化，只有属性需要序列化

小结



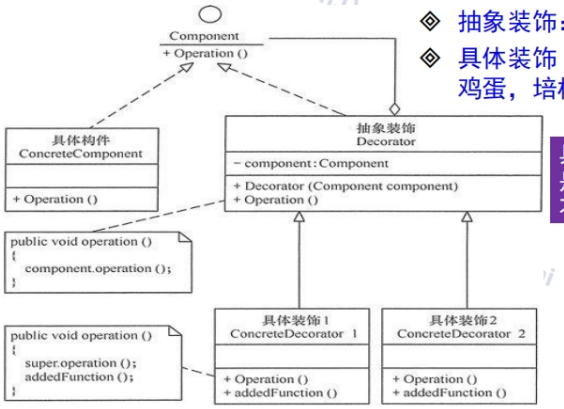
装饰器设计模式

> 装饰器设计模式是一种设计模式，用于在不改变对象自身的基础上，动态地给对象添加额外的职责或功能。这种设计模式提供了比继承更有弹性的替代方案，可以在运行时动态地扩展一个对象的功能。

- 假设有一个餐厅出售快餐
    - 大类：米饭(Rice)，面条(Noodle),...
    - 配菜装饰：蛋炒(EggDeco)，培根(BaconDeco)
    - 每种快餐都有方法：
        - `getPrice()`返回不同价格
        - `cook()`
- 如何设计类间关系

设计范式：装饰器

- 抽象构件（component）：food类；
- 具体构件（ConcreteComponent）：炒饭，炒面



- 抽象装饰：（Decorator）；
- 具体装饰（ConcreteDecorator）：鸡蛋，培根

具体构件共享装饰器，只是装饰类的构造器中对象不同

JAVA

```java
abstract class FastFood {
        private int price;
        public abstract void cook();
        public FastFood(int price){
                this.price = price;
        }
        public int getPrice() {return price;}
}
class Noodle extends FastFood{
        public Noodle(int price) {
                super(price);
        }
        public int getPrice() {return super.getPrice();}
        public void cook() {
                System.out.println("cook noodles");
        }
}
class Rice extends FastFood{//炒饭类
        public Rice(int price) {
                super(price);
        }
        public int getPrice() {return super.getPrice();}
        public void cook() {
                System.out.println("fried rice");
        }
}

abstract class SideDish extends FastFood{ //配菜类
        private FastFood fastFood;  // 组合
        public SideDish(FastFood fastFood, int price) {
                super(price);
                this.fastFood = fastFood;
        }
        public int getPrice() {
                return (fastFood.getPrice() + super.getPrice());
        }
        public void cook() {
                fastFood.cook();
        }
}
class EggDeco extends SideDish {
        public EggDeco(FastFood fastFood, int price) {
                super(fastFood, price);
        }
        public void cook() {
                super.cook();
                System.out.println("add fried eggs");
        }
}
class BalconDeco extends SideDish {
```

```
        public BalconDeco(FastFood fastFood, int price) {
                super(fastFood, price);
        }
        public void cook() {
                super.cook();
                System.out.println("add salty balcon");
        }
}
public class TestDecorator {
        public static void main(String[] args) {
                FastFood rice = new Rice(10);//一开始点了一碗炒饭
                rice.cook();
                System.out.println("price for rice :" + rice.getPrice());
                System.out.println("------------------------");
                FastFood noodle = new Noodle(8);//一开始点了一碗面条
                noodle.cook();
                System.out.println("price for noodle :" + noodle.getPrice());
                System.out.println("------------------------");
                FastFood eggDeco = new EggDeco(rice, 2);//加个蛋
                eggDeco.cook();
                System.out.println("price for rice with fried egg:" +
eggDeco.getPrice());
                System.out.println("------------------------");
                BalconDeco balcony = new BalconDeco(rice, 3);//加个肠
                balcony.cook();
                System.out.println("price for rice with salty balcon:" +
balcony.getPrice());
                System.out.println("------------------------");
                eggDeco = new EggDeco(noodle, 2);//加个蛋
                eggDeco.cook();
                System.out.println("price for noodle with fried egg:" +
eggDeco.getPrice());
                System.out.println("------------------------");
                balcony = new BalconDeco(noodle, 3);//加个肠
                balcony.cook();
                System.out.println("price for noodle with salty balcon:" +
balcony.getPrice());
        }
}
```

基本操作：继承抽象类/接口+组合

JAVA

```java
public abstract class Decorator implements Component {
    protected Component component;

    public Decorator(Component component) {
        this.component = component;
    }

    public void operation() {
        component.operation();
        // 可以添加额外的操作
    }
}
```

注意：

以下哪些属于File类的功能？

a）改变当前目录
b）返回根目录的名字
c）删除文件
d）读取文件中的数据

答案：b c

课堂小问题(2)

以下哪些类的构造方法具有InputStream类型的参数？

a) BufferedInputStream
b) DataInputStream
c) SequenceInputStream
d) FileInputStream

答案：a，b，c

a) `BufferedInputStream`：是的，`BufferedInputStream` 的构造方法可以接受 `InputStream` 类型的参数，用于包装另一个输入流并提供缓冲功能。

b) `DataInputStream`：是的，`DataInputStream` 的构造方法可以接受 `InputStream` 类型的参数，用于包装另一个输入流，并提供用于读取基本数据类型的方法。

c) `SequenceInputStream`：是的，`SequenceInputStream` 的构造方法可以接受两个 `InputStream` 类型的参数，用于合并这两个输入流。

d) `FileInputStream`：不是。`FileInputStream` 的构造方法接受 `String` 或 `File` 类型的参数，用于指定要打开的文件路径或文件对象，而不是直接接受 `InputStream` 类型的参数。

下面代码运行结果描述正确的是

```
public class TestFile {
  public static void main(String s[]) {
    File f = new File("chp_11/OO.txt");
  }
}
```

答案：c

a) 创建文件 chp_11/OO.txt
b) Windows系统中运行出错，路径分割符错误
c) 如果文件不存在，不会创建文件chp_11/OO.txt
d) 假如文件chp_11/OO.txt存在，抛出异常

FileInputStream

从方向上来讲，它是_____流

从数据读取单位来讲，它是_____流

从功能上讲，它是_____流

答案：输入，字节，文件输入