1. 使用**java编译器(javac.exe)**编译源文件，得到字节码文件；

使用**java解释器(java.exe)**来解释字节码文件。

1. **面向对象的重要思想：**通过抽象得到类。

**核心思想：**封装性。即将数据和对数据的操作封装在一起。

**写类的目的：**根据抽象描述一类事物共有的属性和功能，即给出用于创建具体实例（对象）的一种数据类型，描述过程由类体实现

**面向对象编程的三个特性：**封装性、继承性、多态性。

1. **类体的内容由什么构成：**变量的声明、方法。
2. 子类通过**方法的重写**可以隐藏从父类继承的方法。
3. 类名**只能**访问静态变量，不能直接访问实例变量，实例变量要通过具体的对象来访问。
4. 实例方法可操作实例成员变量和静态成员变量，静态方法**只能**操作静态成员变量。
5. **类声明的变量称作对象。类的定义**包括两部分：类声明，类体。
6. **abstract类(抽象类)**：

定义：用关键字abstract修饰的类称为abstract类（抽象类）。

用关键字abstract修饰的方法称为abstract方法（抽象方法）。

特点：对于abstract方法，只允许声明，不允许实现（没有方法体），而且

不允许使用final和abstract同时修饰一个方法或类。

如果一个abstract类是abstract类的子类，它可以**重写**父类的abstract方法，也可以**继承**这个abstract方法。

1. **关键词的访问权限**：
2. **共有变量和共有方法：**

定义：用public修饰的成员变量和方法被称为共有变量和共有方法。

访问权限：当我们在任何一个类中用Tom创建了一个对象后，该对象能访问自己的public变量和类中的public方法。

class Tom{

public float weight;

public float f(float a,float b){

return a+b;

}

}

class Jerry{

void g(){

Tom cat = new Tom();

cat.weight = 23f; //合法

float sum = cat.f(3,4); //合法

}

}

1. **私有变量和私有方法：**

定义：用关键字private修饰的成员变量和方法称为私有变量和私有方法。

访问权限：当我们在另一个类中用类Tom创建了一个对象后，该对象不能访问自己的私有变量和私有方法。

class Tom{

private float weight;

private float f(float a,float b){

return a+b;

}

}

class Jerry{

void g(){

Tom cat = new Tom();

cat.weight = 23f; //非法

float sum = cat.f(3,4); //非法

}

}

1. **友好变量和友好方法：**

定义：不用private、public、protected修饰的成员变量和方法被称为友好变量和友好方法。

访问权限：当在另一个类中用类Tom创建了一个对象后，如果这个类与Tom在同一个包中，那么该对象能访问自己的友好变量和友好方法。在任何一个与Tom同一包的类中，也可以通过Tom类的类名访问Tom类的友好成员变量和友好方法。

class Tom{

float weight;

float f(float a,float b){

return a+b;

}

}

class Jerry{

void g(){

Tom cat = new Tom();

cat.weight = 23f; //合法

float sum = cat.f(3,4); //合法

}

}

1. **受保护的成员变量和方法：**

定义：用protected修饰的成员变量和方法被称为受保护的成员变量和受保护的方法。

访问权限：当在另一个类中用类Tom创建了一个对象后，如果这个类与Tom在同一个包中，那么该对象能访问自己的protected变量和protected方法。在任何一个与Tom同一包的类中，也可以通过Tom类的类名访问Tom类的protected变量和protected方法。

class Tom{

protected float weight;

protected float f(float a,float b){

return a+b;

}

}

class Jerry{

void g(){

Tom cat = new Tom();

cat.weight = 23f; //合法

float sum = cat.f(3,4); //合法

}

}

1. **内部类**：java支持在一个类中声明另一个类，这样的类称为内部类

**匿名类**：java允许我们直接使用一个类的子类的类体创建一个子类对象，也

就是说，创建子类对象时，除了使用父类的构造方法外还有类体，

此类体被认为是一个子类去掉类声明后的类体，称为匿名类。

**异常类**：所谓异常就是程序运行时可能出现的一些错误，异常处理将会改变程序的控制流程，让程序有机会对错误做出处理。

**外嵌类：**包含内部类的类称为内部类的外嵌类。

1. **接口：**定义：使用关键字interface来定义一个接口，接口的定义和类的定

义很相似，分为接口的声明和接口体。

特点：接口包含接口声明和接口体，和类不同的是，接口使用关键字

interface来声明自己是一个接口。

接口体中包含常量的声明（没有变量）和抽象方法两部分。接

口体中只有抽象方法，没有普通的方法，而且接口体中所有的

常量的访问权限一定都是public、所有抽象方法的访问权限一

定都是public。

1. **非抽象类实现接口**必须重写该接口的所有方法，由于接口中的方法一定是public、abstract方法，所以非抽象类在重写接口方法时不仅要去掉abstract修饰给出方法体，而且方法的访问权限一定要明显地用public来修饰。
2. **abstract类和接口的区别：**
3. abstract类和接口都可以有abstract方法。
4. 接口中只可以有常量，不能有变量；而abstract类中即可以有常量也可以有变量。
5. abstract类中也可以有非abstract方法，接口不可以。
6. **私有变量和私有方法**：用关键字private修饰的成员变量和方法称为私有变量和私有方法。
7. **断言语句：**断言语句一般用于程序不准备通过捕捉异常来处理的错误。

断言语句有以下2种格式：

assert booleanExpression;

assert booleanExpression:messageException;

1. **上转型对象：**

定义：假设，A类是B类的父类，当用子类创建一个对象，并把这个对象的引用放到父类的对象中时，例如：

A a;

a = new B();

或

A a;

B b = new B();

a = b;

这时，称对象a是对象b的上转型对象。

特点：

1. 上转型对象不能操作子类新增的成员变量（失去了这部分属性）；不能调用子类新增的方法（失去了一些功能）。
2. 上转型对象可以访问子类继承或隐藏的成员变量，也可以调用子类继承的方法或子类重写的实例方法。上转型对象操作子类继承的方法或子类重写的实例方法，其作用等价于子类对象去调用这些方法。因此，如果子类重写了父类的某个实例方法后，当对象的上转型对象调用这个实例方法时一定调用了子类重写的实例方法。

注：（1）不要将父类创建的对象和子类对象的上转型对象混淆。

（2）可以将对象的上转型对象再强制转换到一个子类对象，这时，该子类对象又具备了子类所有属性和功能。

（3）不可以将父类创建的对象的引用赋值给子类声明的对象。

1. **try-catch：**java使用try-catch语句来处理异常，将可能出现的异常操作

放在try-catch语句的try部分，当try部分中的某个方法调用发生异常后，try部分将立刻结束执行，而转向执行相应的catch部分；所以程序可以将发生异常后的处理放在catch部分。try-catch语句可以由几个catch组成，分别处理发生的相应异常。

try{

包含可能发生异常的语句

}

catch(ExceptionSubClass1 e){

...

}

catch(ExceptionSubClass2 e){

...

}

1. **标识符的命名规则**
2. 标识符由字母、下划线、美元符号和数字组成，长度不受限制。
3. 标识符的第一个字符不能是数字字符。
4. 标识符不能是关键字（关键字有：abstract assert boolean break byte case catch char class const continue default do double else enum extends final finally float for goto if implements import instanceof int interface long native new package private protected public return short static strictfp super switch synchronized this throw throws transient try void volatile while）
5. 标识符不能是true、false和null（尽管true、false和null不是java关键字）。
6. **运算符的优先级**

|  |  |  |  |
| --- | --- | --- | --- |
| 优先级 | 描述 | 运算符 | 结合性 |
| 1 | 分隔符 | [] () . , ; |  |
| 2 | 对象归类，自增自减运算，逻辑非 | instanceof ++ -- ! | 右到左 |
| 3 | 算数乘除运算 | \* / % | 左到右 |
| 4 | 算术加减运算 | + - | 左到右 |
| 5 | 移位运算 | >> << >>> | 左到右 |
| 6 | 大小关系运算 | < <= > >= | 左到右 |
| 7 | 相等关系运算 | == != | 左到右 |
| 8 | 按位与运算 | & | 左到右 |
| 9 | 按位异或运算 | ^ | 左到右 |
| 10 | 按位或 | | | 左到右 |
| 11 | 逻辑与运算 | && | 左到右 |
| 12 | 逻辑或运算 | || | 左到右 |
| 13 | 三目条件运算 | ? : | 左到右 |
| 14 | 赋值运算 | = | 右到左 |

1. **Radom AccessFile—-将一个文本文件倒置输出 （随机流p187）**

import java.io.\*;

public class Nine {

public static void main (String[] args) {

RandomAccessFile inAndOut = null;

int data[] = {1,2,3,4,5,6,7,8,9,10};

try{

inAndOut = new RandomAccessFile("tom.dat","rw");

for (int i = 0; i < data.length; i++) {

inAndOut.writeInt(data[i]);

}

for (long i = data.length-1; i >= 0; i--) {

inAndOut.seek(i\*4);

System.out.printf("\t%d",inAndOut.readInt());

}

inAndOut.close();

}

catch (IOException E){

}

}

}

1. **String类:**
2. 构造字符串对象：

I．常量对象：用双引号括起的字符序列，“你好”、“12.79”。

II．字符串对象:

String s;

s = new String("we are students");

String tom = new String(s);

char a[] = {'J','a','v','a'};

String s = new String(a);

||

String s = new String("Java");

char a[] = {'零','一','二','三','四','五','六','七','八'};

String s = new String(a,2,4);

||

String s = new String("二三四五");

1. String类的常用方法：

I. public int length()：获取一个字符串的长度。

II.public boolean equals(String s)：

比较当前字符串对象的实体是否与参数s指定的字符串的实体相同。

III.public boolean startWith(String s)、public boolean endsWith(String s)：

调用startWith(String s)方法判断当前字符串对象的前缀是否是参数s指定的字符串；调用endsWith(String s)方法判断当前字符串对象的后缀是否是参数s指定的字符串；

IV．public int compareTo(String s)：

按字典序与参数s指定的字符串比较大小。如果当前字符串对象与s相同，该方法返回值为0；如果当前字符串对象大于s，该方法返回正值；如果小于s，返回负值。

V．public boolean contains(String s)：

判断当前字符串对象是否含有参数指定的字符串s。

VI．public int indexOf(String s)：

字符串调用indexOf(String s)从当前字符串的头开始检索字符串s，并返回首次出现s的引位置。如果没有检索到字符串s，该方法返回-1。字符串调用indexOf(String s，int startpoint)方法从当前字符串的startpoint位置处开始检索字符串s，并返回首次出现s的索引位置，如果没有检索到字符串s，该方法返回-1。字符串调用lastIndexOf(String s)方法从当前字符串的头开始检索字符串s，并返回最后出现s的索引位置，如果没有检索到字符串s，该方法返回-1。

VII．public String substring(int startpoint)：

字符串调用该方法获得一个当前字符串的子串，该子串是从当前字符串的startpoint处截取到最后得到的字符串。字符串调用substring(int start，int end)方法获得一个当前字符串的子串，该子串是从当前字符串的start索引位置截取到end索引位置得到的字符串，但不包括end索引位置上的字符。

VIII．public String trim()

一个字符串s通过调用trim()得到一个字符串对象，该字符串对象是s去掉前后空格后的字符串。

1. 字符串与基本数据的相互转化：

将“数字”字符组成的字符串转化为相应的基本数据类型：

public static byte parseByte(String s) throws NumberFormatException

public static short parseShort(String s) throws NumberFormatException

public static long parseLong(String s) throws NumberFormatException

public static float parseFloat(String s) throws NumberFormatException

public static double parseDouble(String s) throws NumberFormatException

将数值转换成字符串：

public static String valueOf(byte n)

public static String valueOf(int n)

public static String valueOf(long n)

public static String valueOf(float n)

public static String valueOf(double n)

1. 对象的字符串表示：一个对象调用toString()方法可以获得该对象的字符串表示。
2. 字符串与字符、字节数组：

public void getChars(int start,int end,char c[],int offset)

字符串调用getChars()方法将当前字符串中的一部分字符复制到参数c指定的数组中，将字符串中从位置start到end-1位置上的字符复制的数组c中，并从数组c的offset处开始存放这些字符。

调用toCharArray()方法将字符串中的全部字符存放在一个字符数组中。

**Calendar类（静态方法）:**Calendar类在java.util包中

1. 使用Calendar类的static方法getInstance()可以初始化一个日历对象。如：Calendar calendar = Calendar.getInstance();
2. calendar对象可以调用方法:

public final void set(int year,int month,int date)

public final void set(int year,int month,int date,int hour,int minute)

public final void set(int year,int month,int date,int hour,int minute,int second)

将日历翻到任何一个时间，当参数year取负数时表示公元前。

1. calendar对象调用方法public int get(int field)可以获取有关年份、月份、小时、星期等信息，参数field的有效值由Calendaar的静态常量指定，例如：

calendar.get(Calendar.MONTH);返回一个整数，如果整数是0表示当前日历是在1月，该整数是1表示当前日历是在2月。

Calendar.get(DAY\_OF\_WEEK)返回一个整数，如果该整数是1表示星期日，如果是2表示星期一，如果是7表示星期六。

日历对象调用：public long getTimeMillis()可以将时间表示为毫秒。

**StringBuffer类:**

1. StringBuffer对象的创建：

构造方法：StringBuffer()

StringBuffer(size)

StringBuffer(String s)

调用capacity()方法可以获取当前实体的实际容量。

1. StringBuffer类的常用方法：

I．append()方法:

将其他java类型数据转化为字符串后，再追加到StringBuffer对象中。

II．public char charAt()和public void setCharAt(int n,char ch):

char charAt(int n)得到参数n指定位置上的单个字符。setCharAt(int n,char ch)将当前StringBuffer对象实体中的字符串位置n处的字符用参数ch指定的字符替换。

III．StringBuffer insert(int index,String str)

StringBuffer对象使用insert方法将参数str指定的字符串插入到参数index指定的位置，并返回当前对象的引用。

IV．public StringBuffer reverse()：

StringBuffer对象使用reverse()方法将该对象实体中的字符翻转，并返回当前对象的引用。

V．StringBuffer delete(int startIndex,int endIndex)：

删除一个子字符串，并返回当前对象的引用，startIndex到endIndex-1的字符串被删除。deleteCharAt(int index)方法删除当前StringBuffer对象实体的字符串中index位置处的一个字符。

VI．StringBuffer replace(int startIndex,int endIndex,String str)

替换一个子字符串， 用str替换startIndex到endIndex-1的字符串，并返回当前对象的引用。

**字节流：**

1. InputStream类：

InputStream类提供的read方法以字节为顺序地读取源中的数据，只要不关闭流，每次调用read方法就顺序地读取源中的其余内容，直到源的末尾或输入流被关闭。

常用方法：

int read()：输入流调用该方法才能够源中读取单个字节的数据，该方法返回字节值（0-255之间的一个整数），如果未读出字节就返回-1.

int read(byte b[])：输入流调用该方法从源中试图读取b.length个字节到b中，返回实际读取的字节数目。如果到达文件的末尾，则返回-1。

int read(byte b[],int off,int len): 输入流调用该方法从源中试图读取len个字节到b中，并返回实际读取的字节数目。如果到达文件的末尾，则返回-1，参数off指定从字节数组的某个位置开始存放读取的数据。

void close()：输入流调用该方法关闭输入流。

long skip(long numBytes)：输入流调用该方法跳过numBytes个字节，并返回实际跳过的字节数目。

1. OutputStream类：

OutputStream类以字节为顺序地写文件，只要不关闭流，每次调用write方法就顺序地向目的地写入内容，直到流被关闭。

常用方法：

void write(int n)： 输出流调用该方法向输出流写入单个字节。

void write(byte b[])：输出流调用该方法向输出流写入一个字节数组。

void write(byte b[],int off,int len)：从给定字节数组中起始于偏移量off处取len个字节写到输出流。

void close()：关闭输出流。

1. **数据类型，精度从低到高的顺序**

byte-short-char-int-long-float-double

1. **正则表达式：**

字符串对象调用public boolean matches(String regex)方法可以判断当前字符串对象是否和参数regex指定的正则表达式匹配。

字符串对象调用public String replaceAll(String regex,String replacement)方法返回一个字符串，该字符串是当前字符串中所有和参数regex指定的正则表达式匹配的子字符串被参数replacement指定的字符串替换后的字符串。

字符串调用public String[] split(String regex)方法时，使用参数指定的正则表达式regex作为分隔标记分解出其中的单词，并将分解出的单词存放在字符串数组中。

1. **开-闭原则：**就是让设计的系统应当对扩展开放，对修改关闭。本质上是指当系统中增加新的模块时，不需要修改现有的模块。
2. **底层容器：**即通常所说的窗口。其他组件必须被添加到底层容器中，以便借助这个底层容器和操作系统进行信息交互。简单的讲，如果应用程序需要一个按钮，并希望用户和按钮交互，即用户单击按钮使程序作出某种相应的操作，那么这个按钮必须出现在底层容器中，否则用户无法看见按钮，更无法让用户和按钮交互。
3. **对话框：**有模式对话框，无模式对话框

如果一个对话框是有模式的对话框，那么当这个对话框处于激活状态时，只让程序响应对话框内部的事件，而且将堵塞其他线程的执行，用户不能再激活对话框所在程序中的其他窗口，直到该对话框消失不可见。

无模式对话框处于激活状态时，能再激活其他窗口，也不堵塞其他线程的执行。

Component—组件，container—容器

**容器和组件的区别：**

1. Java把Component类的子类或间接子类创建的对象称为一个组件。
2. Java把Container的子类或间接子类创建的对象称为一个容器。
3. 可以向容器添加组件。Container类提供了一个public方法add(),一个容器可以调用这个方法将组件添加到该容器中。
4. 容器调用removeAll()方法可以移掉容器中的全部组件；调用remove(Component c)方法可以移掉容器中参数c指定的组件。
5. 容器本身也是一个组件，因此可以把一个容器添加到另一个容器中实现容器的嵌套。
6. **一个数如果恰好等于它的因子之和，这个数就称为“完数”。编写一个应用程序求1000之内的所有完数。**

public class Ten {

public static void main(String args[]) {

int sum=0,i,j;

for(i=1;i<=1000;i++) {

for(j=1,sum=0;j<i;j++) {

if(i%j==0)

sum=sum+j;

}

if(sum==i)

System.out.println("完数:"+i);

}

}

}

1. **1+2+3+4+…+n<250,求n**

public class Ten{

public static void main (String[] args) {

int n = 1;

int i = 1;

int sum = 0;

for (i = 1; i <= n; i++) {

sum = sum+i;

if(sum>250)

break;

n++;

}

System.out.println("满足条件的最大整数:"+(n-1));

}

}

1. **编写一个应用求满足1+2！+3！+…+n!<=9876的最大整数n。**

public class Ten {

public static void main(String args[]) {

int n=1,i=1,jiecheng=1;

long sum=0;

while(true) {

jiecheng=1;

for(i=1;i<=n;i++){

jiecheng=jiecheng\*i;

}

sum=sum+jiecheng;

if(sum>9876)

break;

n++;

}

System.out.println("满足条件的最大整数:"+(n-1));

}

}

1. **设置一个窗口，进行简单的计算**

**MainClass.java**

public class MainClass{

public static void main (String[] args) {

ComputerFrame frame;

frame=new ComputerFrame();

frame.setTitle("算术测试");

frame.setBounds(100,100,650,180);

}

}

**ComputerFrame.java**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class ComputerFrame extends JFrame{

JMenuBar menubar;

JMenu choiceGrade;

JMenuItem grade1,grade2;

JTextField textOne,textTwo,textResult;

JButton getProblem,giveAnwser;

JLabel operatorLabel,message;

Teacher teacherZhang;

ComputerFrame(){

teacherZhang=new Teacher();

teacherZhang.setMaxInteger(20);

setLayout(new FlowLayout());

menubar=new JMenuBar();

choiceGrade=new JMenu("选择级别");

grade1=new JMenuItem("幼儿级别");

grade2=new JMenuItem("儿童级别");

grade1.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent e){

teacherZhang.setMaxInteger(10);

}

});

grade2.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent e){

teacherZhang.setMaxInteger(50);

}

});

choiceGrade.add(grade1);

choiceGrade.add(grade2);

menubar.add(choiceGrade);

setJMenuBar(menubar);

textOne=new JTextField(5);

textTwo=new JTextField(5);

textResult=new JTextField(5);

operatorLabel=new JLabel("+");

operatorLabel.setFont(new Font("Arial",Font.BOLD,20));

message=new JLabel("你还没有回答呢");

getProblem=new JButton("获取题目");

giveAnwser=new JButton("确定答案");

add(getProblem);

add(textOne);

add(operatorLabel);

add(textTwo);

add(new JLabel("="));

add(textResult);

add(giveAnwser);

add(message);

textResult.requestFocus();

textOne.setEditable(false);

textTwo.setEditable(false);

getProblem.setActionCommand("getProblem");

textResult.setActionCommand("answer");

giveAnwser.setActionCommand("answer");

teacherZhang.setJTextField(textOne,textTwo,textResult);

teacherZhang.setJLabel(operatorLabel,message);

getProblem.addActionListener(teacherZhang);

giveAnwser.addActionListener(teacherZhang);

textResult.addActionListener(teacherZhang);

setVisible(true);

validate();

setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);

}

}

**Teacher.java**

import java.util.Random;

import java.awt.event.\*;

import javax.swing.\*;

public class Teacher implements ActionListener{

int numberOne,numberTwo;

String operator="";

boolean isRight;

Random random;

int maxInteger;

JTextField textOne,textTwo,textResult;

JLabel operatorLabel,message;

Teacher(){

random=new Random();

}

public void setMaxInteger(int n){

maxInteger=n;

}

public void actionPerformed(ActionEvent e){

String str=e.getActionCommand();

if(str.equals("getProblem")){

numberOne=random.nextInt(maxInteger)+1;

numberTwo=random.nextInt(maxInteger)+1;

double d=Math.random();

if(d>=0.5)

operator="+";

else

operator="-";

textOne.setText(""+numberOne);

textTwo.setText(""+numberTwo);

operatorLabel.setText(operator);

message.setText("请回答");

textResult.setText(null);

}

else if(str.equals("answer")){

String answer=textResult.getText();

try{

int result=Integer.parseInt(answer);

if(operator.equals("+")){

if(result==numberOne+numberTwo)

message.setText("你回答正确");

else

message.setText("你回答错误");

}

else if(operator.equals("-")){

if(result==numberOne-numberTwo)

message.setText("你回答正确");

else

message.setText("你回答错误");

}

}

catch(NumberFormatException ex){

message.setText("请输入数字字符");

}

}

}

public void setJTextField(JTextField...t){

textOne=t[0];

textTwo=t[1];

textResult=t[2];

}

public void setJLabel(JLabel...label){

operatorLabel=label[0];

message=label[1];

}

}