**面向对象程序设计的主要优点**1.可提高程序的重用性2.可控制程序的复杂性3.可改善程序的可维护性4.能够更好地支持大型程序设计5.增强了计算机处理信息的范围6.能很好地适应新的硬件环境**输入：**Console.WriteLine("Hello, World!"); **输出：**Console.ReadLine(a)

**C#中的数据类型分为两个基本类别 值类型** 表示实际数据**，**只是将值存放在内存中**，**值类型都存储在堆栈中int、char**引用类型**表示指向数据的指针或引用包含内存堆中对象的地址。若为 null，则表示未引用任何对象 类、接口、数组、字符串。**列：**using System;class TestSwitch{public static void Main(){int days = 0; int month = 3;switch(month){case 1:case 3:case 5:case 7:case 8:case 10:case 12: days = 31;break;case 2:days = 28;break;case 4:case 6:case 9:case 11:days = 30;break;default: days = 0;break;}Console.WriteLine("This month has {0} days",days);}}class Child{ public static int minAge = 0; //静态变量 public int age; //公有变量private string name; //私有变量private string sex; //私有变量// 构造函数public Child(string name, int age,string sex){this.name = name;this.age = age;this.sex=sex;}//公有方法public void PrintChild() { Console.WriteLine("{0}, {1} ,{2}岁.", name,sex,age);}}**构造函数具有以下基本特点：** 1) 每个类至少有一个构造函数。若程序代码中没有构造函数则系统会自动提供一个默认的构造函数。 2) 一个构造函数必须和它的类名相同。3) 构造函数不包含任何返回值。class Program{public Program() { Console.WriteLine("null"); }public Program(string str) { Console.WriteLine("string str={0}",str); } public Program(int x) { Console.WriteLine("int x={0} ",x); }static void Main(){ Program aa = new Program();Program bb = new Program("How are you!");Program cc = new Program(10);Console.ReadLine();}}运行结果：nullHow are you!int x=10**定义方法的注意事项**方法可以有参数，也可以没有参数，但是小括号都是必需的。若参数有多个，用逗号分开。 无返回值时，必须声明方法的返回值类型为void。如果声明一个非void类型的方法，则方法中必须至少有一个return语句并返回一个值。 **方法重载：**方法重载是指具有相同的方法名，但签名不同的多个方法可以同时出现在一个类中。当编译一个重载方法的调用时，编译器会自动找到最佳匹配自变量的方法，否则报错。class Program{ public static int Add(int i, int j) { return i + j; }public static string Add(string s1, string s2) { return s1 + s2; } public static long Add(long x) { return x + 5; }static void Main() { Console.WriteLine(Add(1, 2)); Console.WriteLine(Add("1", "2")); Console.WriteLine(Add(10));Console.ReadLine(); }}运行结果：3。12。15**属性与方法的区别**属性不使用括号，但方法一定要使用括号。2) 属性不能指定参数，方法可以指定参数。3) 属性不能使用void类型，方法则可以使用void类型。 4)属性使用方法与变量相同。**This关键字** this引用的是当前实例。this关键字是一个隐含引用，它隐含于每个类的成员函数中。this关键字引用类的当前对象，成员通过this关键字可以知道自己属于哪一个实例。类和对象定义和使用举例**设计一个公司雇员类(employee)，需要存储这些雇员的姓名name、编号（编号初始值缺省为1000)，并设计以下功能：**为姓名name定义属性，设置其get访问器和set访问器。实现无参构造方法，完成从键盘上输入雇员姓名的功能。实现方法，完成从键盘上重新设置编号初始值的功能(setEmployeeNo)。实现方法显示雇员编号和姓名(displayStatus)。class employee{ private string name; //姓名 private int individualEmpNo; //个人编号private static int employeeNo=1000; //雇员编号目前最大值public string Name{ get{return name; }set{ if(value!=""){name=value; } } }public employee() //构造方法{ Console.Write("请输入一个雇员的姓名：");name=Console.ReadLine();individualEmpNo=employeeNo++;}static public void setEmployeeNo() { Console.Write("请输入雇员编号目前的初始值：");employeeNo=Convert.ToInt32(Console.ReadLine()); } public void displayStatus() //显示人员信息{ Console.WriteLine(“编号：”+individualEmpNo+“，姓名："+name);}} class Program{ static void Main(string[]args){employee.setEmployeeNo();employeeemployee1=newemployee();employee1.displayStatus();employeeemployee2=new employee();employee2.displayStatus(); Console.Write("请重新输入最后一个雇员的姓名：");employee2.Name=Console.ReadLine();employee2.displayStatus();Console.ReadLine();}}**声明一个C#派生类的一般格式为：**class 派生类名: 基类名{//派生类新增的数据成员和成员函数}其中，“基类名”是一个已经定义的类的名称，“派生类名”是继承原有类的特性而生成的新类的名称。**重载：**重载指在同一个类中至少有两个方法用同一个名字，但有不同的参数。重载使得从外部来看，一个操作对于不同的对象有不同的处理方法。调用时，根据参数的不同来区别调用哪个方法。方法的返回类型可以相同或不同，但它不足以使返回类型变成唯一的差异。重载方法的参数表必须不同。 **虚函数：**使用虚函数与重写函数时，**需要注意下面几个方面：**1) 虚方法不能声明为静态（static）的。因为静态的方法是应用在类这一层次的，而多态性只能在对象（实例）上运作。2) virtual不能和private一起使用。因为声明为private就无法在派生类中重写了。3) 重写方法的名称、参数个数、类型以及返回值都必须和虚方法的一致。4)虚方法在派生类中不是必须被重写的，不重写时，派生类将自动调用虚方法在基类中的版本。**如果想让方法myMethod被派生类重写，可以使用修饰符virtual声明：**public virtual void myMethod(){……//程序代码}派生类则用override重写:public override void myMethod(){……//程序代}例 利用虚方法多态性编写汽车信息管理程序：class Vehicle //定义车基类{ protected int passengers; //乘客数protected int wheels; //轮子个数protected float weight; //重量protected string videntifier;//标志public Vehicle(int w, float g, int p, string id){ wheels = w;weight = g;videntifier = id;passengers = p; } public virtual void Speak() { } public void Print(){ Console.WriteLine("轮子数：" +wheels); Console.WriteLine("重量："+weight);Console.WriteLine("乘客数：" +passengers); Console.WriteLine("标志：" +videntifier); }}**抽象类：**至少带有一个抽象方法的类是抽象类。抽象类的主要作用是通过它为一个类族建立一个公共的接口，使它们能够更有效地发挥多态特性。抽象类声明了一系列派生类的公共接口，而接口的完整实现，即抽象方法的函数体，要由派生类自己定义。抽象类派生出新的类之后，如果派生类给出所有抽象方法的函数实现，这个派生类就可以声明自己的对象，不再是抽象类，可以实例化；反之，如果派生类没有给出全部抽象函数的实现，这时的派生类仍然是一个抽象类。 抽象类不能实例化。**抽象方法的声明形式如下**:abstract 方法类型 方法名(参数表);

如：abstract void show();（2）**抽象方法在派生类中被实现的形式为：**public override 返回类型 方法名(){…… //方法体}如：public override void show(){…… //方法体}抽象类与非抽象类的主要不同：第一是抽象类不能直接被实例化，只能在派生类中通过继承使用。第二是抽象类可以包含抽象成员，而非抽象类不能包含抽象成员。当从抽象类派生非抽象类时，这些非抽象类必须具体实现所继承的所有抽象成员。abstract class Shape{ public virtual void Draw(){ Console.WriteLine("画一种图形!");}public abstract void Rotate();//抽象方法}class Square : Shape{ public override void Draw(){ Console.WriteLine("画一个正方形!"); }public override void Rotate(){ Console.WriteLine("顺时针方向旋转正方形!"); }}抽象方法的声明与实现形式抽象方法的声明形式如下:abstract 方法类型 方法名(参数表); 如：abstract void show();**（2）抽象方法在派生类中被实现的形式为：**public override 返回类型 方法名(){ …… //方法体}**如：**public override void show(){…… //方法体}抽象类的声明形式abstract class 类名{……public abstract 返回值类型 抽象方法名();……}利用抽象类多态性编写汽车信息管理程序abstract class Vehicle //定义车基类{protected int passengers; //乘客protected int wheels; //轮子个数protected float weight; //重量protected string videntifier;//标志 public Vehicle(int w, float g, int p, string id){ wheels = w;weight = gvidentifier = id;passengers = p;}

public abstract Speak(); //抽象方法public void Print(){ Console.WriteLine("轮子数：" +wheels);Console.WriteLine("重量："+weight);Console.WriteLine("乘客数：" +passengers);

Console.WriteLine("标志：" +videntifier); }}密封类：不能被继承的类，称为密封类。

声明密封类：使用sealed修饰符**[例] 使用密封类**sealed class SealedClass{public double x = 0 , y= 0;public SealedClass(double a, double b){ x=a;y=b;}publicvoidDisplay(){Console.WriteLine(“x = {0} , y = {1}”,x , y);}}public class anotherclass: SealedClass //错误，不能继承密封类{…… //实现代码 }**Connection对象的常用成员**string ConnectionString { get; set; } ：该属性用来获取或设置用于打开SQL Server数据库的字符串。 int ConnectionTimeout { get; } ：该属性用来获取尝试建立连接时终止尝试，并生成错误之前所等待的时间。 string Database { get; } ：该属性用来获取当前数据库或连接打开后要使用的数据库的名称。string DataSource { get; } ：该属性用来设置要连接的数据源实例名称，例如SQL Server的Local服务实例。 public abstract void Open(); 打开数据库连接 public abstract void Close(); 关闭数据库连接

ADO.NET对象的方法：1)SQL Server使用**SqlConnection**对象。SqlConnection connect = new SqlConnection(connectstring);**使用OleDbConnection对象。** DbConnection connect = new OleDbConnection(connectstring); **使用Command对象执行删除数据操作** static void Main(string[] args) { string myConStr = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=Northwind.mdb;"; OleDbConnection myCon = new OleDbConnection(myConStr);myCon.StateChange+=newSystem.Data.StateChangeEventHandler(MyCon\_StateChange); myCon.Open(); OleDbCommand cmd = new OleDbCommand(); cmd.Connection = myCon;

cmd.CommandText = "Delete From Orders Where OrderID < 10250"; int num = cmd.ExecuteNonQuery(); MessageBox.Show("删除的数据行数量为" + num.ToString());  myCon.Close(); } **重要属性 1**Connection属性用来获得或设置该Command对象的连接数据源。 eg:某SqlConnection类型的conn对象连在SQL Server服务器上，又有一个Command类型的**2**对象cmd，可以通过cmd.Connection = conn来让cmd在conn对象所指定的数据库上操作。 string CommandText { get; set; }cmd.CommandText = "Select \* From 表名 Where 表达式 Order by 字段名";**3**根据CommandType属性的不同取值，CommandText属性获取或设置SQL语句、数据表名或存储过程。CommandType CommandType { get; set; }CommandType属性用来获得或设置CommandText属性中的语句是SQL语句、数据表名还是存储过程。cmd.CommandType = CommandType.Text; cmd.CommandText = "SELECT \* FROM Table1";**说明CommandText属性的值是一个SQL语句; DataAdapter对象：** DataSet对象和数据源之间的桥梁，用于从数据源中检索数据、填充 DataSet对象中的表及对DataSet对象做出的更改提交回数据源 SqlDataAdapter OleDbDataAdapter OdbcDataAdapter OracleDataAdapter 它使用Command对象从数据源中检索数据并将更改提交回数据源，Command对象体现在DataAdapter的SelectCommand、InsertCommand、UpdateCommand 和 DeleteCommand 属性 **重要属性** public OleDbCommand SelectCommand { get; set; } 对应的Command对象的CommandText是一个Select语句 public OleDbCommand InsertCommand { get; set; }对应的Command对象的CommandText是一个Insert语句 public OleDbCommand UpdateCommand { get; set; }对应的Command对象的CommandText是一个Update语句 public OleDbCommand DeleteCommand { get; set; } 对应的Command对象的CommandText是一个Delete语句 **DataSet对象1. 数据集 1)**相当于数据库的一个子集，表示相关表、约束和表间关系在内的数据集合**2)**不区分数据库的差别，不同类型DataAdapter的Fill()都填充DataSet类型**填充数据集 调用DataAdapter对象的Fill方法。3．访问数据集**直接使用对象访问数据集中的数据。DataSet对象的数据表集合Tables，DataTable对象的数据行集合Rows、数据列集合Columns。 **Windows窗体**private void button1\_Click(object sender, System.EventArgs e) {}**创建 Graphics 对象的方法：**（1）利用窗体或控件的 Paint 事件中 PaintEventArgs建立图形对象的引用。例如： void Form1\_Paint(object sender, PaintEventArgs e) {Graphics g = e.Graphics;} （2）利用窗体或某个控件的 CreateGraphics 方法建立 Graphics 对象的引用，所建对象是该控件或窗体的绘图区域。例如： Graphics g=this.CreateGraphics(); （3）利用Graphics类的的FromHwnd方法，建立Graphics对象的引用，所建对象是该控件或窗体的绘图区域 （4）利用Graphics类的FromImage静态方法建立Graphics对象，该方法通常用来操纵图形文件。例如：Image img=Image.FromFile(“g1.jpg”);Graphics g=Graphics.FromImage(img);创建画笔对象：1.创建画笔对象Pen pen1=new Pen(Color.Green)**单色画刷(SolidBrush)** 将某一特定区域填入单一的颜色。 SolidBrush类属于System.Drawing名称空间。构造函数：SolidBrush(Color.Color);例如：**线性渐变画刷**SolidBrush bh1 = new SolidBrush(Color.Red)private void menuItem11\_Click(object sender, System.EventArgs e){Graphicsg=pictureBox1.CreateGraphics();g.Clear(this.BackColor);Penp1=new Pen(Color.Green);LinearGradientBrush，bb1=new LinearGradientBrush(newPoint(0,20),newPoint(20,0),Color.Red,Color.Green);g.FillRectangle(bb1,0,0,200,100);bb1.WrapMode=WrapMode.TileFlipX;g.FillRectangle(bb1,0,120,200,100);}

|  |  |
| --- | --- |
| **名称** | **说明** |
| Color | 设置画笔对象的颜色。 |
| DashCap | 设置用在点划线终点样式。 |
| DashStyle | 设置画笔绘制的虚线的线型。 |
| EndCap | 设置画笔绘制的直线终点的样式。 |
| PenType | 获取画笔绘制的直线的样式。 |
| StartCap | 设置画笔绘制的直线起点的样式。 |
| Width | 设置画笔的宽度。 |

1. **定义类：Person；**a）增加公有成员变量：姓名；b）增加成员属性：性别（公共只读）c）增加成员属性：年龄（派生类可读）d）增加成员属性：职业（公共可读可写）e）增加成员属性：收入（派生类可读可写）f）增加成员函数：MyCareer()，函数功能：输出姓名、年龄、收入等信息；2）定义Person类的派生类Employee3）Employee类中增加函数ChangeInfo()，尝试读写基类Person的成员（变量和属性），体会不同访问限定符的作用；4）在主程序类中实例化Person和Employee，访问Person类的成员。注：不同的访问限定符情况下，程序编译时可能会提示出错，要求截取不同情况下编译器出错提示，并分别说明出错原因usingSystem;usingSystem.Collections.Generic;usingSystem.Linq;usingSystem.Text;

namespace ConsoleApplication1{class Program{static void Main(string[] args){Person p = new Person();

Employee e = new Employee();p.Name = "EDG";p.Sex = 0;p.Job = "doctor";p.Age = 20;p.Income = 6666;}}class Person{public string Name;public int Sex{get; private set;}protected int Age{get;private set;}

public string Job{get;set;}protected int Income{get;set;}public void MyCareer(){Console.WriteLine("name" + Name);Console.WriteLine("age" + Age);Console.WriteLine("income" + Income)；Console.WriteLine("sex" + Sex);Console.WriteLine("job" + Job);}}class Employee : Person

{void ChangeInfo(){Name = "Tony";Age = 32;Income = 30000;Sex = 0;Job = "FBI";}}}**1、实现类Student**1）定义类Student及其成员（姓名，学号，年龄和成绩）；2）为类Student添加实例成员函数Info()，该函数具有输出姓名，学号，年龄和成绩等信息的功能；3）为主程序类添加静态成员函数StudentInfo(Student stud)，该函数具有输出stud的姓名，学号，年龄和成绩等信息的功能；4）在主程序中实例化Student；5）在主程序类的Main函数中调用Student类的Info函数显示实例信息；6）在主程序类的Main函数中调用StudentInfo函数显示实例信息；

**1.实现类Student，**using System;using System.Collections.Generic;usingSystem.Linq;

using System.Text;namespace ConsoleApplication2{class Program{static void StudentInfo(Student stud){stud.Info();}static void Main(string[] args)

{Student stud = new Student();stud.Info();Program.StudentInfo(stud);}}

class Student{public string Name;string SeriesNo;int Age;double Score;

public Student(){this.Name = "zou";this.SeriesNo = "33";this.Age = 19;this.Score = 100; }public void Info(){Console.Write("姓名：" + Name + ";");Console.Write("学号：" + SeriesNo + ";");Console.Write("年龄：" + Age + ";");Console.Write("分数：" + Score + ";");Console.ReadLine();}}}**2、实现静态类TriangleOperator**1）创建三角形计算方法静态类TriangleOperator2）类中添加静态成员函数：计算周CalaPerimeter(double edge1, double edge2, double angle)；周长公式：p = a + b + c已知两边及其夹角求第三边1665858421508230003722.png1665858421508230003722.png3）类中添加静态成员函数：计算面积，CalcArea(double edge1, double edge2, double angle)；面积公式：area = 7403530161508230003784.png三角函数（sin\cos）及平方根函数(sqrt)在静态类Math中using System;using System.Collections.Generic;using System.Linq;using System.Text;namespace ConsoleApplication1{class Program{

static void Main(string[] args){Triangle tri = new Triangle(3, 4, 90);

Console.WriteLine(tri.perimeter);}}class Triangle{private double edge1, edge2, angle;public Triangle(int x, int y){edge1 = x;edge2 = y;angle = 30 \* Math.PI / 180;}public Triangle(int x, int y, int a){edge1 = x;edge2 = y;angle = a \* Math.PI / 180;}public double perimeter{get { return TriangleOperator.CalaPerimeter(edge1, edge2, angle); }}}static class TriangleOperator{public static double CalaPerimeter(double edge1, double edge2, double angle){double edge3 = Math.Sqrt(edge1 \* edge1 + edge2 \* edge2 - 2 \* edge1 \* edge2 \* Math.Cos(angle));return edge1 + edge2 + edge3; }}1、完成教材P221的图10.56对应程序功能。private void Form1\_Paint(object sender, PaintEventArgs e){Graphics g = e.Graphics;Pen p = new Pen(Color.Blue, 2);g.DrawLine(p, 10, 10, 100, 100);g.DrawRectangle(p, 10, 10, 100, 100); g.DrawEllipse(p, 10, 10, 100, 100);}**2、绘制圆形、正方形**1）编辑框输入数字控制绘制图形大小；2）单选框控制当前绘制图形类型。3）点击窗体客户区，以鼠标点击位置为中心绘制圆形、正方形private void Form1\_MouseDown(object sender, MouseEventArgs e){ int r = Convert.ToInt32(textBox1.Text);int ShapType = 0;if (radioButton1.Checked)

ShapType = 1;elseShapType = 2;Graphics g = this.CreateGraphics(); if (ShapType == 1)g.DrawEllipse(new Pen(Color.Blue), e.X - (int)r / 2, e.Y - (int)r / 2, (int)r, (int)r);elseg.DrawRectangle(new Pen(Color.Blue), e.X - (int)r / 2, e.Y - (int)r / 2, (int)r, (int)r);}}**3、颜色拾取**1）构建窗体应用程序，添加PictureBox控件和TextBox控件；2）选择图片文件，显示在PictureBox控件中；3）点击PictureBox，获取鼠标位置，并获得图片对应位置的像素，将像素的颜色值（r，g，b）显示在TextBox中。

private void pictureBox1\_MouseDown(object sender, MouseEventArgs e){Bitmap bmp = new Bitmap(pictureBox1 .Image );Color clr = bmp.GetPixel(e.X, e.Y);textBox1.Text = "R:" + clr.R.ToString() + " G:" + clr.G.ToString() + " B:" +clr.B.ToString();}