2\_4如何定义静态方法，静态方法有何特点？

用static修饰的方法是仅属于类的静态方法。与此同时，不用static修饰的方法，则为实例方法。

静态方法的本质是该方法是属于整个类的，不是属于某个实例的；静态方法效率上要比实例化高，但静态方法的缺点是不自动进行销毁，这可能会造成内存资源的浪费，而实例化的则可以做销毁；静态方法和静态变量创建后始终使用同一块内存空间，而使用实例的方式则会创建多个内存空间。

2\_8 什么是访问控制符?有哪些访问控制符?哪些可以用来修饰类?哪些用来修饰域和方法?试述不同访问控制符的作用。

访问控制符是一组限定类、域或方法是否可以被程序里的其他部分访问和调用的修饰符。

C# 中的访问控制符有5个，其中基本的有4个：public,protected,private,internal，还有一个复合的修饰符protected internal（也可以写成internal protected）。

类的访问控制符只有一个public，域和方法的访问控制符有五个，分别是public、private、protected、internal、protected internal。

3\_1

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_3\_1

{

class Program

{

static void Main(string[] args)

{

Student stu = new Student();

stu.setAge(20);

Console.WriteLine(stu.getAge());

Console.ReadKey();

}

}

public class Student

{

//域（Field），就是字段

//编写一个C#程序定义一个表示学生的类student，包括域“学号”“班号”“姓名”“性别”“年龄”；方法“获得学号”“获得班号”“获得性别”“获得年龄”“修改年龄”。

public string SN;

public string getSN(){ return SN; }

public string CN;

public string getCN() { return CN; }

public string Name;

public string getName() { return Name; }

public string Sex;

public string getSex() { return Sex; }

public double Age;

public double getAge() { return Age; }

public void setAge(double v) { Age=v; }

}

}

3\_2

using System;

namespace \_3\_2\_综合练习\_编写银行ATM程序

{

class Account

{

//Constructor

Account() { }

Account(long bank\_cardID, string password)

{

this.bank\_cardID = bank\_cardID;

this.account\_password = password;

}

//fields

private static long initial\_ID = 1000000001; //the 1st one to create an account get this ID number

private static string bank\_name = "ICBC";

private long bank\_cardID;

public string account\_password;

private long total\_amount = 100000; //initial account

private string[] data = new string[5];

private string[] keys =

{

"card ID","holder's name", "total sum", "latest withdraw","latest deposit"

};

//property

public long latest\_withdraw { set; get; }

public long latest\_deposit { set; get; }

public string date\_withdraw { set; get; }

public string date\_deposit { set; get; }

public string date\_create { set; get; }

//indexer

public string this[int i]

{

set

{

data[i] = value;

}

get

{

if (i >= 0 && i < data.Length)

return data[i];

return null;

}

}

public string this[string key]

{

get

{

return this[FindIndex(key)];

}

}

private int FindIndex(string key)

{

for (int i = 0; i < keys.Length; i++)

if (keys[i] == key)

return i;

return -1;

}

//methods

//withdraw from the account, record the current time

public void withdrawMoney()

{

Console.Write("amount(withdraw): ");

latest\_withdraw = Convert.ToInt32(Console.ReadLine());

if (latest\_withdraw <= total\_amount)

{

total\_amount -= latest\_withdraw;

this[2] = Convert.ToString(total\_amount);

date\_withdraw = DateTime.Now.ToString();

this[3] = Convert.ToString(latest\_withdraw);

}

else

Console.WriteLine("Lack of balance. Operation is refused\n");

}

//deposit from the account, record the current time

public void depositMoney()

{

Console.Write("amount(deposit): ");

latest\_deposit = Convert.ToInt32(Console.ReadLine());

if (latest\_deposit > 0)

{

total\_amount += latest\_deposit;

this[2] = Convert.ToString(total\_amount);

date\_deposit = DateTime.Now.ToString();

this[4] = Convert.ToString(latest\_deposit);

}

else

Console.WriteLine("Invalid operation\n");

}

//get information about the account

void get\_card\_info() //try 4 choices below

{

Console.WriteLine("( card ID / holder's name / total sum / latest withdraw / latest deposit )?");

string instr = Console.ReadLine();

if (instr == "card ID" || instr == "holder's name" || instr == "total sum" || instr == "latest withdraw"

|| instr == "latest deposit")

{

this[3] = Convert.ToString(latest\_withdraw);

this[2] = Convert.ToString(total\_amount);

Console.Write(instr + " is " + this[instr]);

if (instr == "latest withdraw")

Console.WriteLine(" " + date\_withdraw);

else if (instr == "latest deposit")

Console.WriteLine(" " + date\_deposit);

else if (instr == "card ID")

Console.WriteLine(" " + date\_create);

else if (instr == "card ID" || instr == "total sum")

Console.WriteLine("\n");

}

else

Console.WriteLine("Invalid input!!");

}

//Inheritance, subclass CreditAccount

protected class CreditAccount : Account

{

//Constructor

CreditAccount(long bank\_cardID, string password)

{

this.bank\_cardID = bank\_cardID;

this.account\_password = password;

}

//new field

private long line\_of\_credit; //line of credit

//new property

public string credit\_rating { set; get; }

//new method

public long get\_line\_of\_credit() //line of credit according to the credit rating

{

if (credit\_rating == "3" || credit\_rating == "2")

line\_of\_credit = 50000;

else if (credit\_rating == "1" || credit\_rating == "0")

line\_of\_credit = 10000;

else

line\_of\_credit = 0;

return line\_of\_credit;

}

//override method withdrawMoney()

new public void withdrawMoney()

{

Console.Write("amount(withdraw): ");

latest\_withdraw = Convert.ToInt32(Console.ReadLine());

if (latest\_withdraw <= total\_amount + line\_of\_credit)

{

total\_amount -= latest\_withdraw;

this[2] = Convert.ToString(total\_amount);

date\_withdraw = DateTime.Now.ToString();

this[3] = Convert.ToString(latest\_withdraw);

if (latest\_withdraw >= total\_amount)

{

Console.WriteLine("warning: you're using your credit!! Withdraw successfully");

int temp = Convert.ToInt32(credit\_rating);

credit\_rating = Convert.ToString(--temp);

get\_line\_of\_credit();

}

}

else

{

Console.WriteLine("Lack of balance. Operation is refused\n");

}

}

public static void Main(String[] args)

{

Account a;

CreditAccount ca;

string card\_category;

//create a new account, set password, get an ID number

void create\_account()

{

Console.WriteLine("######### " + bank\_name + " #########"); //which bank

Console.Write("create an account ( normal / credit )?");

card\_category = Console.ReadLine();

if (card\_category != "credit" && card\_category != "normal")

{

Console.WriteLine("Invalid input");

create\_account();

}

Console.Write("set password: ");

string password = Console.ReadLine(); //set password

Account a\_create = new CreditAccount(initial\_ID, password);

a = a\_create;

ca = (CreditAccount)a;

a[0] = Convert.ToString(initial\_ID); //save ID

Console.Write("Your name: ");

a[1] = Console.ReadLine(); //save owner's name

a[2] = Convert.ToString(a.total\_amount);

a.date\_create = DateTime.Now.ToString(); //save the time that this account was created

Console.WriteLine("create successfully!!\nYour ID: " + initial\_ID + " " +

"Remember your password:" + password + " You have $100000 initially.");

initial\_ID++;

a.latest\_deposit = 0;

a.latest\_withdraw = 0;

if (card\_category == "credit")

{

ca.credit\_rating = "3";

ca.get\_line\_of\_credit();

}

}

create\_account();

while (true)

{

if (card\_category == "normal")

{

//ask for the next instruction from the user

Console.WriteLine("( create again / get information / withdraw / deposit )?");

switch (Console.ReadLine())

{

case "create again": create\_account(); break;

case "get information":

a.get\_card\_info(); break;

case "withdraw":

a.withdrawMoney();

a[2] = Convert.ToString(a.latest\_withdraw);

break;

case "deposit":

a.depositMoney();

a[3] = Convert.ToString(a.latest\_deposit);

break;

default:

Console.WriteLine("invalid input\n");

break;

}

}

else if (card\_category == "credit")

{

//ask for the next instruction from the user

Console.WriteLine("( create again / get information / withdraw / deposit / line of credit )?");

switch (Console.ReadLine())

{

case "create again": create\_account(); break;

case "get information":

ca.get\_card\_info(); break;

case "withdraw":

ca.withdrawMoney();

ca[2] = Convert.ToString(ca.latest\_withdraw);

break;

case "deposit":

ca.depositMoney();

ca[3] = Convert.ToString(ca.latest\_deposit);

break;

case "line of credit":

Console.WriteLine("LIne of credit: " + ca.get\_line\_of\_credit());

break;

default:

Console.WriteLine("invalid input\n");

break;

}

}

}

}

}

}

}