# 浙江理工大学 2018—2019 学年第 2 学期

# 《C#程序设计》期末试卷（A）卷

本人郑重承诺：本人已阅读并且透彻地理解《浙江理工大学考场规则》，愿意在考试中自觉遵守这些规定，保证按规定的程序和要求参加考试，如有违反，自愿按《浙江理工大学学生违纪处分规定》有关条款接受处理。

**承诺人签名：_____ 学号：_____ 班级：_____**

**一、判断题（判断下列各题是否正确，正确的在题后的括号里打 √，错误的打×。每小题 1 分，共 10 分。）**

(1) const 只能用于基本类型及 string ( )

(2) 所有的对象都有 ToString()方法 ( )

(3) Person p1 = new Person(18); //Person 是引用类型
Person p2 = p1;
如果 p1.age=35;
则 p2.age 也是 35 ( )

(4) abstract 表示抽象的，不能被实例化，也就是说不能有构造方法 ( )

(5) Lambda 表达式的函数参数类型不可以省略 ( )

(6) 运算符本质上是一个函数，但是书写起来更直观 ( )

(7) out 参数在函数中必须赋值后才能返回 ( )

(8) String 对象的内容是不可变的 ( )

(9) is 运算符用于判断运行时对象的类型 ( )

(10) [Serialize]这个 Attribute 表示对象可序列化 ( )

**二、单选题（在每小题的四个备选答案中选出一个正确答案，并将正确答案的序号填入题后的括号内。每小题 2 分，共 20 分。）**

(1) 调用方法结束后，（　　）不再存在。
A. 值传递的形式参数及其值
B. 引用传递的实际参数及其值
C. 用 ref 修饰的参数及其值
D. 用 out 修饰的参数及其值

(2) 在定义类时，如果希望类的某个方法能够在派生类中进一步进行改进，以处理不同的派生类的需要，则应将该方法声明成（　　）。
A．sealed 方法　　B．public 方法　　C．virtual 方法　　D．override 方法

(3) 以下（　　）类的对象是 ADO.NET 在非连接模式下处理数据内容的主要对象。
A．Command　　B．Connection　　C．DataAdapter　　D．DataSet

(4) 用 FileStream 对象打开一个文件时，可用 FileMode 参数控制（　　）。
A．对文件覆盖、创建、打开等选项中的哪些操作
B．对文件进行只读、只写还是读写
C．其他 FileStream 对象对同一个文件所具有的访问类型
D．对文件进行随机访问时的定位点

(5) 以下泛型集合声明中正确的是（　　）。
A．List<int> f = new List<int>();　　B．List<int> f = new List();
C．List f = new List();　　D．List<int> f = new List<int>;

(6) 在 C#中，以下关于属性的描述正确的是（　　）。
A．属性是以 public 关键字修饰的字段，以 public 修饰的字段也可称为属性
B．属性是访问字段的一种灵活机制，属性更好地实现了数据的封闭和隐藏
C．要定义只读属性只需在属性名前加上 readonly 关键字
D．在 C#中类中不能自定义属性

(7) 以下 C#程序的执行情况是（　　）。

```
using System;
namespace aaa
{
    delegate void delep(int i);
    class Program
    {   static void Main()
        {   funb(new delep(funa));    }
        public static void funa(int t)
        {   funb(21);    }
        public static void funb(int i)
        {   Console.WriteLine(i.ToString());   }
    }
}
```

A．代码中存在错误，delegate void delep(int i);不能定义在名称空间或者类之外
B．代码中存在错误，代码行 funb(new delep(funa));使用委托错误
C．程序正常运行，输出为 0
D．程序正常运行，输出为 21

(8) 在 C#中接口与抽象基类的区别在于（　　）。
A．抽象类可以包含非抽象方法，而接口不包含任何方法的实现
B．抽象类可以被实例化，而接口不能被实例化
C．抽象类不能被实例化，而接口可以被实例化
D．抽象类能够被继承，而接口不能被继承

(9) 关于委托的说法，不正确的是（　　）。
A. 委托属于引用类型　　B. 委托用于封装方法的引用
C. 委托可以封装多个方法　　D. 委托不必实例化即可被调用

(10) 以下关于泛型的叙述中错误的是（　　）。
A．泛型是通过参数化类型来实现在同一份代码上操作多种数据类型
B．泛型编程是一种编程范式，其特点是参数化类型
C．泛型类型和普通类型的区别在于泛型类型与一组类型参数或类型变量关联
D．以上都不对

## 三、程序设计题（本题共 70 分）

1、（共 27 分）已定义接口 IBankAccount 和类 POS（表示店里支付用的 pos 机）

```
interface IBankAccount
{
    bool Pay(POS pos, double amount);
    double Balance { get; }
}
```

```
class POS{
    public string Area { get; set; }
    public POS(string area)
    {
        Area = area;
    }
}
```

按要求编写以下类：
1）定义一个类 SaveAccount，表示一个储蓄账户。它继承 IBankAccount，有构造方法，实现接口功能，其中支付时不得透支。如有必要可添加字段。[8 分]
2）定义一个类 LocalAccount，表示一个本地储蓄账户。它继承 SaveAccount，有属性 Area（办卡地），有构造方法，其中在外地支付时，需额外支付 2%。[7 分]
3）定义一个类 CreditAccount，表示一个信用卡账户。它继承 SaveAccount，有属性 Limit（透支额度），有只读属性 IsOverdraw（是否透支），有构造方法，其中支付时不超过透支额度就可以。[9 分]
4）在主方法中对上述功能进行测试。[3 分]

2、（共 25 分）写一个时间类 Time，具有：
1）时、分、秒属性，并保证不超出正常时间范围（从 0:0:0 到 23:59:59）。[8 分]
2）无参构造函数和三参数构造函数。[4 分]
3）写一个 bool TryParse(String str, out Time t)方法，对格式类似“2:15:30”的字符串进行解析。如果不超出正常时间范围，则输出该时间，返回 true，否则返回 false。[7 分]。提示：string[] strs = str.Split(″:″)可将字符串分隔为几个子串，″:″为分隔符。
4）重载“-”（减号）运算符，使两个 Time 实例相减，返回 int 值，表示两时刻相差的秒数。[4 分]

5）重载 ToString 方法，将时分秒信息显示出来。[2 分]

3、（共 18 分）使用委托、Lambda 表达式、LINQ 等知识完成以下内容

（1）定义一个枚举类型 Gender，包含 Male 和 Female 两个值。[2 分]

（2）要求定义 Customer 类，带有：ID 属性（int 类型）、Name 属性（string 类型）、Birthday 属性（DateTime 类型）、Gender 属性（Gender 类型）。[2 分]

（3）写一个静态方法 FindCustomer，根据 filter 参数在数组中查找符合条件的 Customer 实例，方法声明为：[5 分]

```
List<Customer> FindCustomers(List<Customer> custs, Predicate<Customer> predicate)
```

（4）写一个 lambda 表达式，赋值为 Predicate<Customer>变量，调用（3）中方法，把男性客户找出来并打印。[5 分]

（5）利用 LINQ 技术进行查询，把女性客户找出来，并按名字排序。[4 分]

主方法中已提供以下代码和数据：

```
public static void Main(string[] argv)
{
            List<Customer> customers = new List<Customer> {
              new Customer { ID = 1001, Name = "Damon", Birthday = new DateTime(1988, 5,
1), Gender = Gender.Male },
              new Customer { ID = 1002, Name = "Niki", Birthday = new DateTime(1995, 10,
4), Gender = Gender.Female },
              new Customer { ID = 1003, Name= "Ayrton", Birthday = new DateTime(1982, 6,
23), Gender = Gender.Male },
              new Customer { ID = 1004, Name= "Graham", Birthday = new DateTime(1994, 9,
15), Gender = Gender.Female }
            };    };
}
```

# 浙江理工大学 2018—2019 学年第 2 学期

# 《C#程序设计》期末试卷（A）卷标准答案和评分标准

一、判断题（本大题共 10 分，每小题 1 分）

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| √ | √ | √ | X | X |
| 6 | 7 | 8 | 9 | 10 |
| √ | √ | √ | √ | X |

一、单选题（本大题共 20 分，每小题 2 分）

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| A | C | D | A | A |
| 6 | 7 | 8 | 9 | 10 |
| B | B | A | D | D |

二、程序设计题（共 70 分）

1、

```
class SaveAccount : IBankAccount//1
{
    private double balance;//1
    public SaveAccount(double balance)//1
    {
        this.balance = balance;//1
    }
    public virtual bool Pay(POS pos, double amount)//1
    {
        if(balance < amount)//1
        {
            Console.WriteLine("Payment failed");
            return false;//1
        }
        else
        {
```

```
            balance -= amount;
            return true;
        }
    }
    public double Balance//1
    {
        get
        {
            return balance;
        }
    }

    class LocalAccount : SaveAccount//1
    {
        public string Area { get; set; }//1
        public LocalAccount(string area, double balance) : base(balance)//2
        {
            Area = area;
        }
        public override bool Pay(POS pos, double amount)//1
        {
            if (pos.Area == this.Area)//1
                return base.Pay(pos, amount);
            else
            {
                double n = (double)(amount * 1.02);//1
                return base.Pay(pos, n);
            }
        }
    }
    class CreditAccount : SaveAccount//1
    {
        public double Limit { get; set; }//1
        public CreditAccount(double balance, double limit) : base(balance)//2
        {
            Limit = limit;
        }
        public bool IsOverdraw//1
        {
            get
            {
                return balance < 0;//1
            }
```

```
        }
        public override bool Pay(POS pos, double amount)//1
        {
            if (balance + Limit < amount)//1
            {
                Console.WriteLine("Withdrawal attempt failed");
                return false;
            }
            else
            {
                balance -= amount;//1
                return true;
            }
        }
    }
}
```

2、

```
    class Time//1
{
    private int hour;//1
    public int Hour//1
    {
        get
        {
            return hour;//1
        }
        set
        {
            if (hour < 0 || hour > 23)//1
                throw new ArgumentException("Hour is not valid!");
            hour = value;//1
        }
    }
    private int minute;//1
    public int Minute
    {
        get
        {
            return minute;
        }
        set
        {
```

```
        if (minute < 0 || minute > 59)
            throw new ArgumentException("Minute is not valid!");
        minute = value;
    }
}
private int second;//1
public int Second
{
    get
    {
        return second;
    }
    set
    {
        if (second < 0 || second > 59)
            throw new ArgumentException("Minute is not valid!");
        second = value;
    }
}
public Time(int hour, int min, int sec)//2
{
    Hour = hour;
    Minute = min;
    Second = sec;
}
public Time() : this(0, 0, 0)//2
{ }
public bool TryParse(string str, out Time t)//1
{
    t = new Time();//1
    if (str == null)
        return false;

    string[] strs = str.Split(":");//1
    if (strs.Length != 3)//1
        return false;

    int h, m, s;
    if (int.TryParse(strs[0], out h)//1
        && int.TryParse(strs[1], out m)
        && int.TryParse(strs[2], out s))
    {
        t = new Time(h, m, s);//1
```

```
                return true;
            }

            return false;//1
        }
        public override string ToString()//1
        {
            return string.Format($"{Hour}:{Minute}:{Second}");//1
        }
        public static int operator - (Time t1, Time t2)//1
        {
            int result = 0;//1
            result += 3600 * (t1.Hour - t2.Hour);//1
            result += 60 * (t1.Minute - t2.Minute);
            result += t1.Second - t2.Second;
            return result;//1
        }
    }
```

3

```
    enum Gender { Male, Female}//2
    class Customer//2
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public Gender Gender { get; set; }
        public DateTime Birthday { get; set; }
        public override string ToString()
        {
            return String.Format($"{ID}:{Name}:{Gender}-{Birthday}");
        }

        public static List<Customer> FindCustomers(List<Customer> custs, Predicate<Customer>
predicate)//1
        {
            List<Customer> customers = new List<Customer>();//1
            foreach(Customer cust in custs)//1
            {
                if (predicate(cust))//1
                    customers.Add(cust);//1
            }
            return customers;
        }
```

```
        public static void Test()
        {
            List<Customer> customers = new List<Customer> {
              new Customer { ID = 1001, Name = "Damon", Birthday = new DateTime(1988, 5, 1),
Gender = Gender.Male },
              new Customer { ID = 1002, Name = "Niki", Birthday = new DateTime(1995, 10, 4),
Gender = Gender.Female },
              new Customer { ID = 1003, Name= "Ayrton", Birthday = new DateTime(1982, 6, 23),
Gender = Gender.Male },
              new Customer { ID = 1004, Name= "Graham", Birthday = new DateTime(1994, 9, 15),
Gender = Gender.Female }
            };

            Predicate<Customer> predicate = x => x.Gender == Gender.Male;//1
            List<Customer> lst = FindCustomers(customers, predicate);//2
            foreach(Customer cust in lst)//2
                Console.WriteLine(cust);

            var query = from cust in customers//1
                        where cust.Gender == Gender.Female//1
                        orderby cust.Name//1
                        select cust;
            foreach (Customer cust in query)//1
                Console.WriteLine(cust);
        }
    }
```