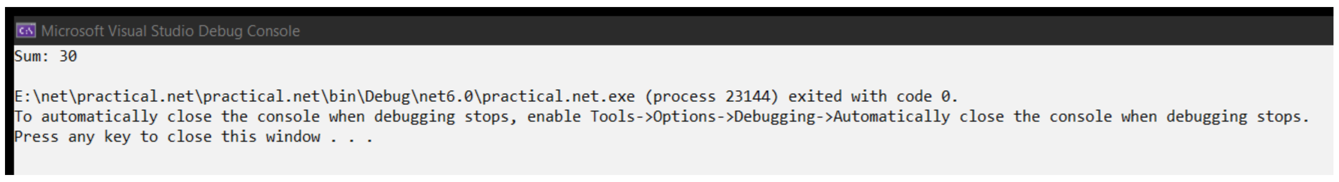
To Show Constructor, Method Overloading and Indexers

Constructor

Code

```
using System;
namespace ParameterizedConstructor
{
        class Sum
        {
             private int x;
             private int y;
             public Sum(int a, int b)
            {
                 x = a;
                 y = b;
            }
            public int getSum()
            {
                 return x + y;
            }
        }
        class Test
        {
             static void Main(string[] args)
             {
                   Sum s = new Sum(20, 10);
                   Console.WriteLine("Sum: {0}", s.getSum());
             }
        }
}
```

Output



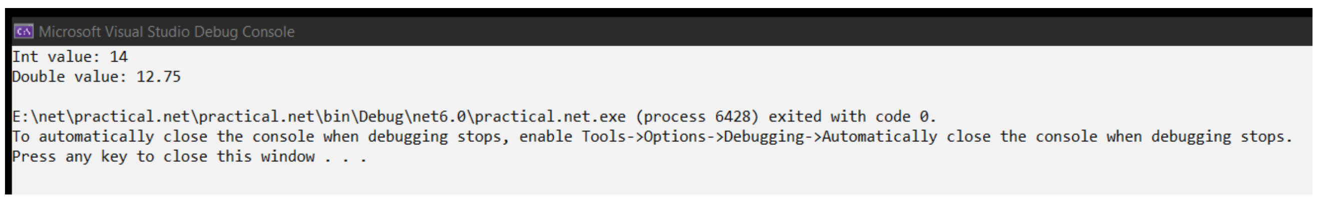
```
Microsoft Visual Studio Debug Console
Sum: 30

E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe (process 23144) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Method overloading

Code

```
namespace MyApplication
{
        class Program
        {
            static int Add(int x, int y)
            {
                 return x + y;
            }
            static double Add(double x, double y)
           {
                 return x + y;
           }
           static void Main(string[] args)
          {
                int myNum1 = Add(5, 9);
                double myNum2 = Add(5.5, 7.25);
                Console.WriteLine("Int value: " + myNum1);
                Console.WriteLine("Double value: " + myNum2);
          }
        }
}
```

Output



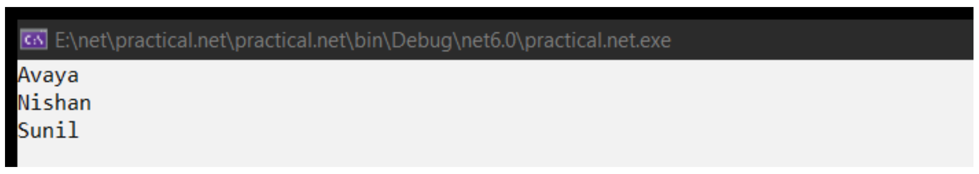
```
Microsoft Visual Studio Debug Console
Int value: 14
Double value: 12.75

E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe (process 6428) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Indexer

Code

```
using System;
namespace Indexer_example
{
   class Program
   {
      class IndexerClass
      {
         private string[] names = new string[10];
         public string this[int i]
         {
            get
            {
               return names[i];
            }
            set
            {
               names[i] = value;
            }
         }
      }
      static void Main(string[] args)
      {
         IndexerClass Team = new IndexerClass();
         Team[0] = "Avaya";
         Team[1] = "Nishan";
         Team[2] = "Sunil";
         for (int i = 0; i < 10; i++)
         {
            Console.WriteLine(Team[i]);
         }
         Console.ReadKey();
      }
   }
}
```

Output



```
E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe
Avaya
Nishan
Sunil
```

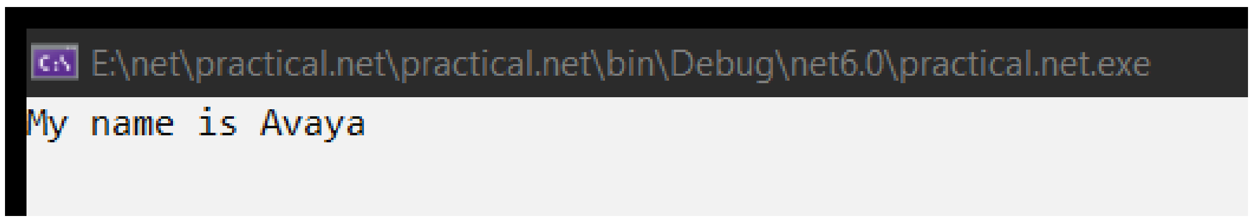
To Show Inheritance, Sealed Class and use of BASE keyword

Inheritance:

Code

```
using System;
namespace Inheritance
{
   class A
   {
      public string name;
   }
   class B : A
   {
      public void getName()
      {
         Console.WriteLine("My name is " + name);
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         B obj = new B();
         obj.name = "Avaya";
         obj.getName();
         Console.ReadLine();
      }
   }
}
```

Output



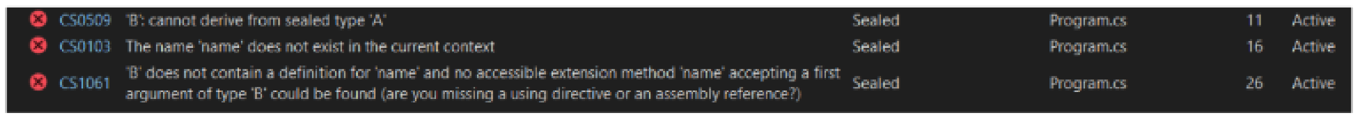
```
E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe
My name is Avaya
```

Sealed Class

Code

```
using System;
using System.Xml.Linq;

namespace Inheritance
{
   sealed class A
   {
      public string name;
   }
   class B : A
   {
      public void getName()
      {
         Console.WriteLine("My name is " + name);
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         B obj = new B();
         obj.name = "Avaya";
         obj.getName();
         Console.ReadLine();
      }
   }
}
```

Output



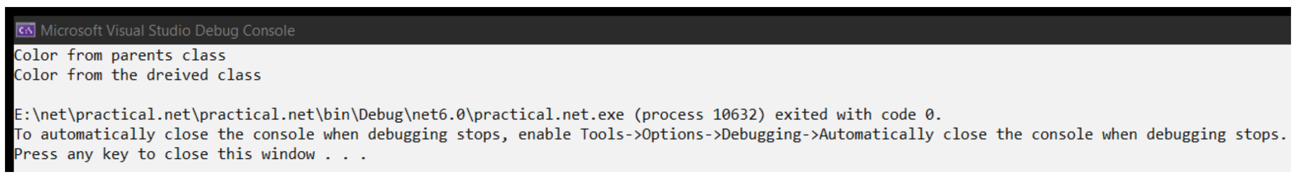
```
CS0509  'B': cannot derive from sealed type 'A'                                      Sealed   Program.cs   11   Active
CS0103  The name 'name' does not exist in the current context                        Sealed   Program.cs   16   Active
CS1061  'B' does not contain a definition for 'name' and no accessible extension method 'name' accepting a first
        argument of type 'B' could be found (are you missing a using directive or an assembly reference?)
                                                                                     Sealed   Program.cs   26   Active
```

Base keyword

Code

```
using System;
public class A
{
   public string color = "Color from parents class";
}
public class B : A
{
   public string color = "Color from the dreived class";
   public void Show()
   {
      Console.WriteLine(base.color);
      Console.WriteLine(color);
   }
}
public class MainClass
{
   public static void Main()
   {
      B obj1 = new B();
      obj1.Show();
   }
}
```

Output



```
Microsoft Visual Studio Debug Console
Color from parents class
Color from the dreived class

E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe (process 10632) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

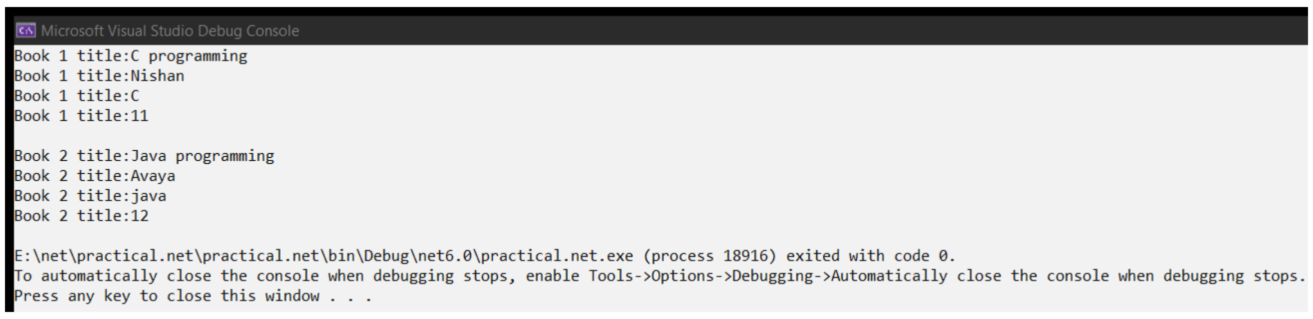
To Show Struct, Enum and Delegates

Struct

Code

```
using System;
namespace Struct
{
    struct Books
    {
        public string title;
        public string author;
        public string subject;
        public int book_id;
    }
    public class testStructure
    {
        public static void Main(string[] args)
        {
            Books Book1;
            Books Book2;
            Book1.title = "C programming";
            Book1.author = "Nishan";
            Book1.subject = "C";
            Book1.book_id = 11;
            Book2.title = "Java programming";
            Book2.author = "Avaya";
            Book2.subject = "java";
            Book2.book_id = 12;
            Console.WriteLine("Book 1 title:{0}", Book1.title);
            Console.WriteLine("Book 1 title:{0}", Book1.author);
            Console.WriteLine("Book 1 title:{0}", Book1.subject);
            Console.WriteLine("Book 1 title:{0}", Book1.book_id);
            Console.WriteLine("\nBook 2 title:{0}", Book2.title);
            Console.WriteLine("Book 2 title:{0}", Book2.author);
            Console.WriteLine("Book 2 title:{0}", Book2.subject);
            Console.WriteLine("Book 2 title:{0}", Book2.book_id);
        }
    }
}
```

Output



```
Microsoft Visual Studio Debug Console
Book 1 title:C programming
Book 1 title:Nishan
Book 1 title:C
Book 1 title:11

Book 2 title:Java programming
Book 2 title:Avaya
Book 2 title:java
Book 2 title:12

E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe (process 18916) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

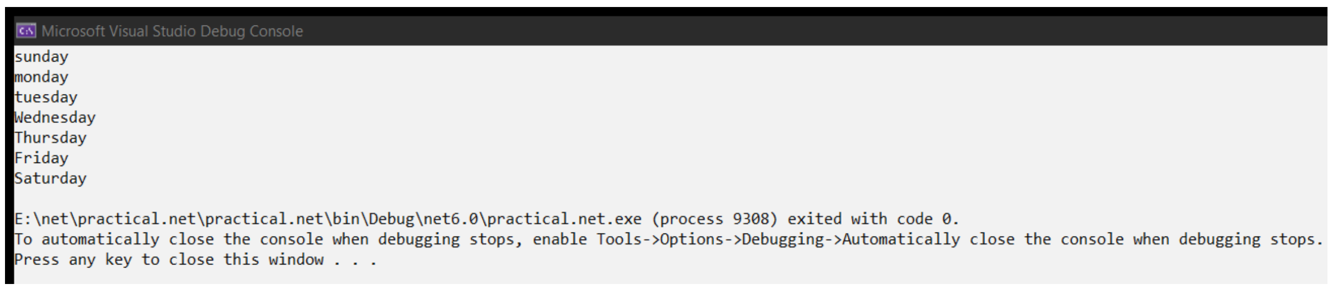
Enum

Code

```
using System;
// define an enum
enum Weekdays
{
   sunday,
   monday,
   tuesday,
   Wednesday,
   Thursday,
   Friday,
   Saturday
}
class Program
{
   public static void Main()
   {
      foreach (Weekdays d in Enum.GetValues(typeof(Weekdays)))
      {
         Console.WriteLine(d);
```

```
        }
    }
}
```

Output



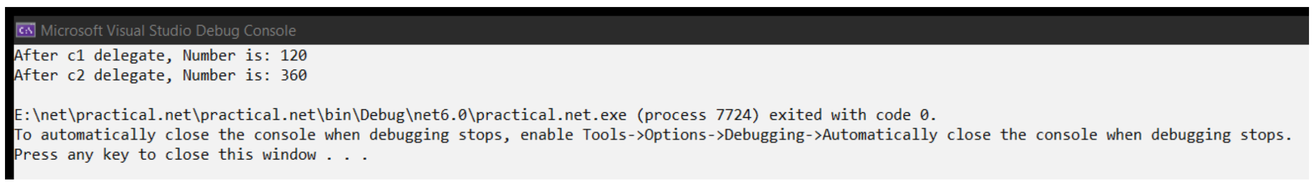
```
Microsoft Visual Studio Debug Console
sunday
monday
tuesday
Wednesday
Thursday
Friday
Saturday

E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe (process 9308) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Delegates

Code

```
using System;
delegate int Calculator(int n);
public class DelegateExample
{
    static int number = 100;
    public static int add(int n)
    {
        number = number + n;
        return number;
    }
    public static int mul(int n)
    {
        number = number * n;
        return number;
    }
    public static int getNumber()
    {
        return number;
    }
    public static void Main(string[] args)
    {
        Calculator c1 = new Calculator(add);
        Calculator c2 = new Calculator(mul);
        c1(20);
        Console.WriteLine("After c1 delegate, Number is: " + getNumber());
        c2(3);
        Console.WriteLine("After c2 delegate, Number is: " + getNumber());
    }
}
```

Output



```
Microsoft Visual Studio Debug Console
After c1 delegate, Number is: 120
After c2 delegate, Number is: 360

E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe (process 7724) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

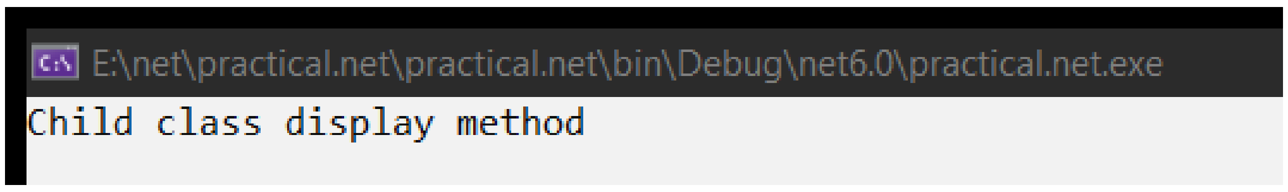
To Show Method Hiding and Method Override

Method Hiding

Code

```
using System;
namespace MethodHiding
{
    class Class1
    {
        public void display()
        {
            Console.WriteLine("Parent class display method");
        }
    }
    class Class2 : Class1
    {
        public new void display()
        {
            Console.WriteLine("Child class display method");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Class2 obj = new Class2();
            obj.display();
            Console.ReadKey();
        }
    }
}
```

Output



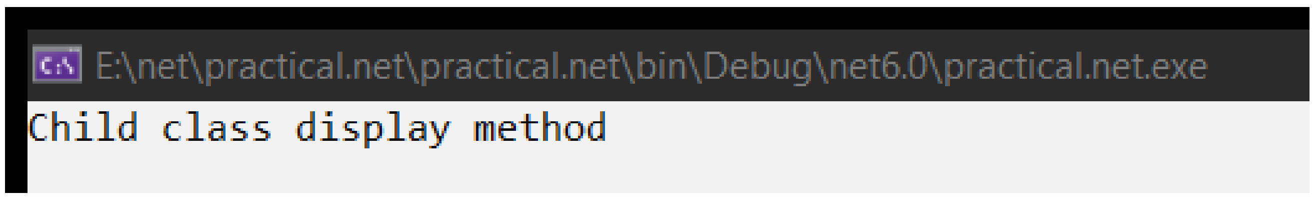
```
E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe
Child class display method
```

Method Overriding

Code

```
using System;
namespace MethodHiding
{
    class Class1
    {
        public virtual void display()
        {
            Console.WriteLine("Parent class display method");
        }
    }
    class Class2 : Class1
    {
        public override void display()
        {
            Console.WriteLine("Child class display method");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Class2 obj = new Class2();
            obj.display();
            Console.ReadKey();
        }
    }
}
```

Output



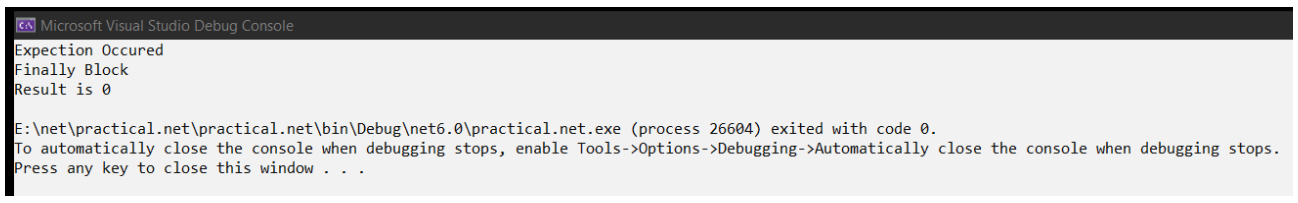
```
E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe
Child class display method
```

To Handle Exceptions in C#

Code

```
using System;
public class ExExample
{
    public static void Main()
    {
        int x = 0;
        int div = 0;
        try
        {
            div = 100 / x;
            Console.WriteLine("This is not executed");
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Expection Occured");
        }
        finally
        {
            Console.WriteLine("Finally Block");
        }
        Console.WriteLine($"Result is {div}");
    }
}
```

Output



```
Expection Occured
Finally Block
Result is 0

E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe (process 26604) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

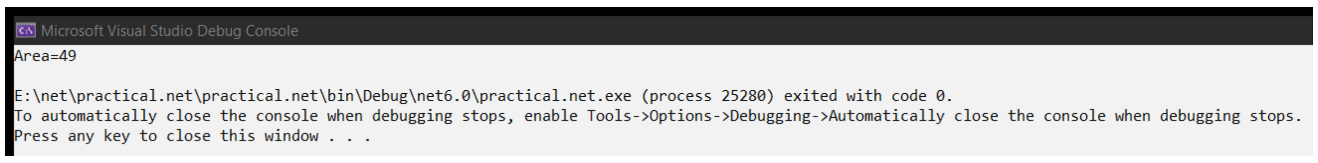
To Show Abstract Classes and Interfaces in C#

Abstract Classes

Code

```
using System;
using System.Security.Principal;
abstract class AreaClass
{
    abstract public int Area();
}
class Square : AreaClass
{
    int side = 0;
    public Square(int n)
    {
        side = n;
    }
    public override int Area()
    {
        return side * side;
    }
}
class Driver
{
    public static void Main()
    {
        Square s = new Square(7);
        Console.WriteLine("Area=" + s.Area());
    }
}
```

Output



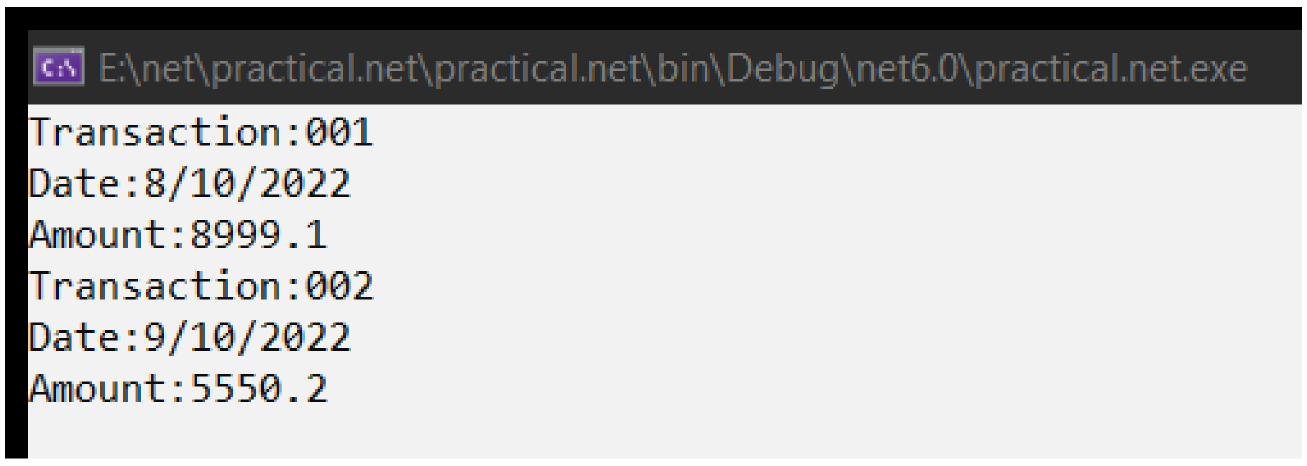
```
Area=49

E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe (process 25280) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Interface

Code

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System;
namespace InterfaceApplication
{
   public interface ITransactions
   {
      void showTransaction();
      double getAmount();
   }
   public class Transaction : ITransactions
   {
      private string tCode;
      private string date;
      private double amount;
      public Transaction()
      {
         tCode = " ";
         date = " ";
         amount = 0.0;
      }
      public Transaction(string c, string d, double a)
      {
         tCode = c;
         date = d;
         amount = a;
      }
      public double getAmount()
      {
         return amount;
      }
      public void showTransaction()
      {
         Console.WriteLine("Transaction:{0}", tCode);
         Console.WriteLine("Date:{0}", date);
         Console.WriteLine("Amount:{0}", getAmount());
      }
   }
   class Tester
   {
      static void Main(string[] args)
      {
         Transaction t1 = new Transaction("001", "8/10/2022", 8999.10);
         Transaction t2 = new Transaction("002", "9/10/2022", 5550.20);
         t1.showTransaction();
         t2.showTransaction();
         Console.ReadKey();
      }
   }
}
```

Output



```
E:\net\practical.net\practical.net\bin\Debug\net6.0\practical.net.exe
Transaction:001
Date:8/10/2022
Amount:8999.1
Transaction:002
Date:9/10/2022
Amount:5550.2
```