I.1. Tham chiếu this

Trong lập trình đơn tuyến với C#, tại mỗi thời điểm chỉ có một hàm được thực thi. Mỗi khi gọi thực thi một phương thức của một đối tượng (không phải là phương thức tĩnh) thì tham chiếu ***this*** tự động trỏ đến đối tượng này. Mọi phương thức của đối tượng đều có thể tham chiếu đến các thành phần của đối tượng thông qua tham chiếu ***this***. Có 3 trường hợp thường dùng tham chiếu ***this***:

- Tránh xung đột tên khi tham số của phương thức trùng tên với tên biến dữ liệu của đối tượng.
- Dùng để truyền đối tượng hiện tại làm tham số cho một phương thức khác (chẳng hạn gọi đệ qui)
- Dùng với mục đích chỉ mục.

Ví dụ 1: Dùng tham chiếu ***this*** với mục đích tránh xung đột tên của tham số với tên biến dữ liệu của đối tượng.

```
public class Date
{
      private int Year;
      private int Month;
      private int Day;

      public Date(int Day, int Month, int Year)
      {
            Console.WriteLine("Constructor co 3 tham so!");
            this.Year = Year;
            this.Month = Month;
            this.Day = Day;
      }
… các phương thức khác
}
```

Ví dụ 2: bài toán tháp Hà Nội, minh họa dùng tham chiếu *this* trong đệ qui.

```
using System;

class Cot
{
      uint [] Disks;
      uint Size, Top = 0;
      char Ten;

      public Cot(uint n, char TenCot)     {
            Size = n;
            Ten = TenCot;
            Disks = new uint[Size];
      }

      public void ThemDia(uint DiskID)    {
            Disks[Top] = DiskID;
```

```
            Top++;
      }

//    ~Cot()     {Console.WriteLine("Freeing {0}", Ten);}

// Chuyen mot dia tren dinh tu cot hien hanh sang cot C
// Ham nay su dung tham chieu this de chuong trinh duoc
ro rang hon
      public void Chuyen1Dia(Cot C) {
            this.Top--;
            Console.WriteLine("Chuyen dia {0} tu {1} sang
            {2}",     this.Disks[Top], this.Ten, C.Ten);

            //C.ThemDia(this.Disks[this.Top]);
            C.Disks[C.Top] = this.Disks[this.Top];
            C.Top++;
      }
// Chuyen N dia (SoDia) tu cot hien hanh sang cot C, lay
cot B lam trung //gian. Ham nay su dung tham chieu this
voi muc dich de qui
      public void ChuyenNhieuDia(uint SoDia, Cot B, Cot
C){
            // Chuyen mot dia sang C
            if( SoDia == 1) Chuyen1Dia(C);
            else      {
                  ChuyenNhieuDia(SoDia-1,C, B);
                  Chuyen1Dia(C);
                  B.ChuyenNhieuDia(SoDia -1,this, C);
            }
      }
}

class This_ThapHaNoiApp
{
      static void Main()  {
            Cot A, B, C;
            uint SoDia = 4;
            A = new Cot(SoDia, 'A');
            B = new Cot(SoDia, 'B');
            C = new Cot(SoDia, 'C');

            //Nap N dia vao cot A
            uint i = SoDia;
            while(i>0)           {
                  A. ThemDia(i);
                  i--;
            }

//Chuyen N dia tu cot A sang cot C, lay cot B lam trung
gian
            A.ChuyenNhieuDia(SoDia,B, C);
```

```
            Console.ReadLine();
        }
}
```

I.2. Đóng gói dữ liệu với thuộc tính (property)

Thuộc tính là một đặc tính mới được giới thiệu trong ngôn ngữ **C#** làm tăng sức mạnh của tính đóng gói.

Thuộc tính đơn giản là các phương thức lấy giá trị (***get***) và gán giá trị (***set***), thường liên quan đến một biến hoặc một đại lượng nào đó của đối tượng. Nó cho phép truy cập đến các thành phần dữ liệu của đối tượng ở mức độ đọc hoặc ghi hoặc cả hai và che dấu cài đặt thực sự bên trong lớp. Một thuộc tính thường quản lý một biến dữ liệu của đối tượng hoặc của lớp. Thuộc tính có một trong ba tính chất sau:

- Chỉ đọc (*read-only*): chỉ có phương thức ***get***. Ta chỉ được đọc giá trị của thuộc tính.
- Chỉ ghi (*write-only*): chỉ có phương thức ***set***. Ta chỉ được gán (ghi dữ liệu) giá trị cho thuộc tính.
- Vừa đọc vừa ghi (*read/write*): có cả hai phương thức ***get*** và ***set***. Được phép đọc và ghi giá trị.

Cú pháp định nghĩa một thuộc tính:

```
public KiểuTrảVề  TênThuộcTính {
     // phương thức lấy giá trị của thuộc tính
     get {
          //…các câu lệnh
          return BiểuThứcTrảVề;
     }

     //phương thức gán giá trị cho thuộc tính
(biến)
     set {
          //…các câu lệnh
          BiếnThànhViên = value;
     }
}
```

Cú pháp định nghĩa một thuộc tính khá giống cú pháp định nghĩa một hàm không có tham số. Phần thân của phương thức *get* của thuộc tính tương tự như phần thân phương thức của lớp. Chúng trả về một giá trị hoặc một biến nào đó, thường là trả về giá trị của biến thành viên mà thuộc tính đó quản lý. Khi ta truy xuất đến thuộc tính để lấy giá trị thì phương thức ***get*** được gọi thực hiện.

Phương thức ***set*** của thuộc tính dùng để gán giá trị cho biến thành viên mà thuộc tính quản lý. Khi định nghĩa phương thức ***set***, ta phải sử dụng từ khóa ***value*** để biểu diễn cho giá trị dùng để gán cho biến thành viên. Khi gán một giá

trị cho thuộc tính thì phương thức ***set*** tự động được gọi và tham số ẩn ***value*** chính là giá trị dùng để gán.

Ví dụ: Định nghĩa lớp Student với 3 thuộc tính tương ứng với điểm toán, điểm tin và

```
using System;

class Student {
      //Chú ý rằng tên của các biến có dấu "_" còn tên của
      //các thuộc tính tương ứng thì không có dấu "_".
      string _Ten ;
      float _DiemToan, _DiemTin;
      float _DiemTB;

      // constructor
      public Student()     {
            _Ten = "";
            _DiemToan = 0;
            _DiemTin = 0;
            _DiemTB = 0;
      }

      // thuoc tinh ten - (read/write)
      public string Ten    {
            get {      return _Ten;    }
            set {      _Ten = value;   }
      }

      //Thuoc tinh diem toan - (read/write)
      public float DiemToan     {
            get {      return _DiemToan;    }
            set {
                  _DiemToan = value;
                  _DiemTB = (_DiemToan + DiemTin)/2;
            }
      }

      //Thuoc tinh diem tin - (read/write)
      public float DiemTin      {
            get {      return _DiemTin;     }
            set
            {
                  _DiemTin = value;
                  _DiemTB = (_DiemToan + DiemTin)/2;
            }
      }

      //Thuoc tinh diem tin - (read only)
      public float DiemTrungBinh     {
            get {      return _DiemTB;      }
```

```
    }
}

class Student_PropertyApp     {
    static void Main(string[] args)    {
        Student s1 = new Student();
        s1.Ten = "Hoa";
        s1.DiemToan = 5;
        s1.DiemTin = 7;
        //s1.DiemTrungBinh = 6;  ---> loi
        Console.WriteLine("Ten:  {0},  diem  Toan:  {1},
        diem Tin: {2}, diem trung binh: {3}", s1.Ten,
        s1.DiemToan, s1.DiemTin, s1.DiemTrungBinh);

        Console.ReadLine();
    }
}
```

Trong ví dụ trên, ***Ten, DiemToan, DiemTin*** là các thuộc tính vừa đọc vừa ghi và ***DiemTrungBinh*** là thuộc tính chỉ đọc. Chúng phủ lên các thành phần dữ liệu tương ứng là ***_Ten, _DiemToan, _DiemTin, _DiemTB***, giúp người thiết kế che dấu cài đặt thực sự bên trong lớp. Các thuộc tính ***DiemToan, DiemTin, DiemTrungBinh*** không cho phép người dùng gán giá trị cho biến ***_DiemTB*** và che dấu cách cài đặt của điểm trung bình.

Khi ta gọi các lệnh:

```
s1.Ten = "Hoa";
s1.DiemToan = 5;
s1.DiemTin = 7;
```

thì phương thức ***set*** của các thuộc tính ***Ten, DiemToan, DiemTin*** tương ứng được gọi và tham số ẩn ***value*** lần lượt là ***"Hoa", 5,*** *7*.

Khi ta gọi các lệnh:

```
Console.Write("Ten: {0}, diem Toan: {1}, ", s1.Ten,
s1.DiemToan);
Console.WriteLine(" diem Tin: {0}, diem TB: {1}",
s1.DiemTin, s1.DiemTrungBinh);
```

thì phương thức ***get*** của các thuộc tính ***Ten, DiemToan, DiemTin, DiemTrungBinh*** tương ứng được gọi.

Nếu ta gọi lệnh:

```
s1.DiemTrungBinh = 6;
```

thì trình biên dịch sẽ báo lỗi vì thuộc tính ***DiemTrungBinh*** không cài đặt phương thức ***set***.

> ***Bài tập 1:*** *Viết chương trình xây dựng lớp TamGiac với dữ liệu là 3 cạnh của tam giác. Xây dựng các thuộc tính (property) ChuVi, DienTich và các phương thức kiểm tra kiểu của tam giác (thường, vuông, cân, vuông cân, đều).*

Bài tập 2: *Viết chương trình xây dựng lớp HinhTruTron (hình trụ tròn) với dữ liệu chiều cao và bán kính. Xây dựng các thuộc tính (property) DienTichDay (diên tích mặt đáy), DienTichXungQuanh (diên tích mặt xung quanh), TheTich (thể tích).*

I.3. Toán tử (operator)

Trong C#, toán tử là một phương thức tĩnh dùng để định nghĩa chồng một phép toán nào đó trên các đối tượng. Mục đích của toán tử là để viết mã chương trình gọn gàng, dễ hiểu hơn, thay vì phải gọi phương thức.

Ta có thể định nghĩa chồng các toán tử sau:

- Toán học: +, -, *, /, %.
- Cộng trừ một ngôi: ++, --, -.
- Quan hệ so sánh: ==, !=, >, <, >=, <=.
- Ép kiểu: ().
- …

Cú pháp khai báo nguyên mẫu của một toán tử T:

public static KiểuTrảVề operator T (CácThamSố)
{
///các câu lệnh trong thân toán tử
}

Ví dụ:

Toán tử cộng 2 phân số cần định nghĩa như sau:

```
public static PhanSo operator+(PhanSo P1, PhanSo P2)
{
        ...
}
```

Một số chú ý:

- Tham số của toán tử phải là kiểu tham trị (không dùng các từ khóa *ref, out*).
- Không được định nghĩa chồng toán tử = (gán), && , || (and, or logic), ?: (điều kiện), checked, unchecked, new, typeof, as, is.
- [] không được xem là một toán tử.
- Khi định nghĩa chồng các toán tử dạng: +, -, *, / , % thì các toán tử +=, -=, *=, /= , %= cũng tự động được định nghĩa chồng.
- Khi định nghĩa chồng toán tử thì nên định nghĩa chồng theo cặp đối ngẫu. Chẳng hạn, khi định nghĩa chồng toán tử == thì định nghĩa chồng thêm toán tử !=.

- Khi định nghĩa chồng toán tử ==, != thì nên định nghĩa thêm các phương thức *Equals()*, *GetHashCode()* để đảm bảo luật *"hai đối tượng bằng nhau theo toán tử == hoặc phương thức Equals sẽ có cùng mã băm"*.
- Khi định nghĩa toán tử ép kiểu ta phải chỉ ra đây là toán tử ép kiểu ngầm định (implicit) hay tường minh (explicit).

Cú pháp định nghĩa toán tử ép kiểu:

public static [implicit | explicit] operator KiểuTrảVềT (Type V)

Ý nghĩa: Chuyển đổi dữ liệu kiểu ***Type V*** sang dữ liệu kiểu ***KiểuTrảVềT***.

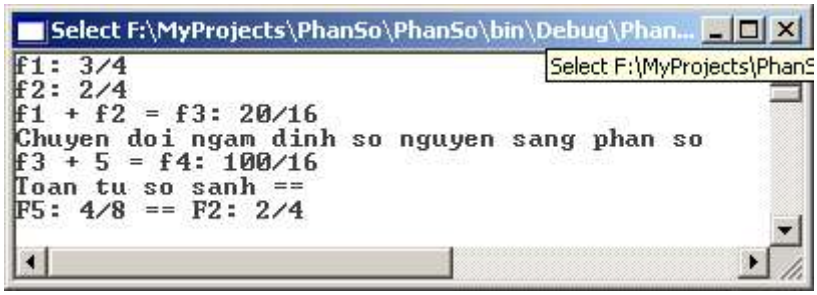
Ví dụ: xây dựng lớp phân số và định nghĩa chồng các phép toán trên phân số.

```
using System;

class PhanSo {
      int Tu, Mau;    // private members

      //constructor
      public PhanSo(int TuSo, int MauSo) {
            Tu = TuSo;
            Mau = MauSo;
      }

      //constructor
      public PhanSo(int HoleNumber) {
            Tu = HoleNumber;
            Mau = 1;
      }

      //constructor
      public PhanSo()      {
            Tu = 0;
            Mau = 1;
      }

      //Chuyen doi ngam dinh tu so nguyen sang phan so
      public static implicit operator PhanSo(int theInt) {
            Console.WriteLine("Chuyen so nguyen thanh phan
            so");
            return new PhanSo(theInt);
      }

      //Chuyen doi tuong minh phan so sang so nguyen;
      public static explicit operator int(PhanSo PS)     {
            return PS.Tu/PS.Mau;
      }

      //toan tu so sanh ==
      public  static  bool  operator==(PhanSo  PS1,  PhanSo
PS2){
```

```
            return (PS1.Tu * PS2.Mau == PS2.Tu * PS1.Mau);
      }

      // Toan tu so sanh !=;
      public static bool operator!=(PhanSo PS1, PhanSo
PS2){
            return !(PS1 == PS2);
      }

      // phong thuc so sanh 2 phan so co bang nhau hay
khong
      public override bool Equals(object o)    {
            Console.WriteLine("Phuong thuc Equals");
            if (! (o is PhanSo) )     return false;
            return this == (PhanSo) o;
      }

      //Toan tu cong hai phan so
      public static PhanSo operator+(PhanSo PS1, PhanSo
PS2) {
            int MauMoi = PS1.Mau * PS2.Mau ;
            int TuMoi = PS2.Mau * PS1.Tu + PS1.Mau * PS2.Tu;
            return new PhanSo(TuMoi, MauMoi);
      }

      // Tang phan so them mot don vi!
      public static PhanSo operator++(PhanSo PS)    {
            PS.Tu = PS.Mau +  PS.Tu;
            return PS;
      }

      //ep phan so ve gia tri True, false de tra loi cau
doi: day co phai la mot phan so hop le hay khong
      public static implicit operator bool(PhanSo PS)    {
            return PS.Mau !=0;
      }

      //Phuong thuc doi phan so thanh chuoi
      public override string ToString()  {
            String   s   =   Tu.ToString(   )   +   "/"   +
Mau.ToString( );
            return s;
      }
}
class PhanSoApp {
      static void Main( ) {
            PhanSo f1 = new PhanSo(3,4);
            Console.WriteLine("f1: {0}", f1.ToString( ));

            PhanSo f2 = new PhanSo(2,4);
            Console.WriteLine("f2: {0}", f2.ToString( ));
```

```
            PhanSo f3 = f1 + f2;
            Console.WriteLine("f1   +   f2   =   f3:   {0}",
f3.ToString());

            PhanSo f4 = f3 + 5;
            Console.WriteLine("f3   +   5   =   f4:   {0}",
f4.ToString( ));

            PhanSo f5 = new PhanSo(4,8);
            if (f5 == f2)  {
                  Console.WriteLine("F5:  {0}  ==  F2:  {1}",
                        f5.ToString( ), f2.ToString( ));
            }
            Console.ReadLine();
      }
}
```

Kết quả của chương trình:



Bài tập 1: *Xây dựng lớp SoPhuc (số phức) với các phương thức và các toán tử +, -, *, / , ép sang kiểu số thực…*
Bài tập 2: *Xây dựng lớp MaTranVuong (ma trận vuông) với các phương thức và các toán tử +, -, *, / , ép sang kiểu số thực (trả về định thức của ma trận)…*

I.4. Indexer (Chỉ mục)

Việc định nghĩa chỉ mục (indexer) cho phép tạo các đối tượng của lớp hoạt động giống như một mảng ảo nhằm cho phép sử dụng toán tử [] để truy cập đến một thành phần dữ liệu nào đó của đối tượng. Chỉ mục không thể là static.

Việc định nghĩa chỉ mục tương tự như việc định nghĩa một thuộc tính.

Cú pháp tạo chỉ mục:

```
public KiểuTraVề this[DanhSáchThamSố]    {
      //Hàm đọc
      get   {
            //thân hàm đọc
      }
      // Hàm ghi
```

```
            set  {
                 //thân hàm ghi
            }
      }
```

Ví dụ 1: *Định nghĩa lớp mảng và dùng indexer để truy cập trực tiếp đến các phần tử của mảng:*

```
using System;
class ArrayIndexer {
     int [] myArray;
     int Size;

     public ArrayIndexer (int n)    {
          Size = n;
          myArray = new int[Size];
     }

// chi muc cho phép truy cập đến phần tử thứ index trong
mảng.
     public int this [int index]    {
          get  {
               // Kiểm tra ngoài mảng hay không
               if (index < 0 || index >= Size)return 0;
               //throw                                  new
IndexOutOfRangeException();
               else return myArray[index];
          }
          set  {
               if (!(index < 0 || index >= Size))
                    myArray[index] = value;
          }
     }
}

public class MainClass {
     public static void Main()      {
          ArrayIndexer b = new ArrayIndexer(10);
          b[3] = 256;
          b[4] = 1024;
          for (int i=0; i<5; i++)
               Console.WriteLine("b[{0}] = {1}", i, b[i]);
                    Console.ReadLine();
     }
}
```

Trong lớp ***ArrayIndexer*** trên ta định nghĩa một chỉ mục trả về kiểu ***int*** và có một tham số ***index*** là chỉ số của phần tử cần truy cập trong mảng.

```
public int this [int index]
```

Phương thức ***get*** lấy giá trị của phần tử thứ ***index*** , phương thức ***set*** gán giá trị cho phần tử thứ ***index*** trong mảng.

Ví dụ 2:

Giả sử ta cần định nghĩa một lớp có nhiệm vụ thao tác trên file như thao tác trên một mảng các byte. Lớp này hữu dụng khi cần thao tác trên một file rất lớn mà không thể đưa toàn bộ file vào bộ nhớ chính tại một thời điểm, nhất là khi ta chỉ muốn thay đổi một vài byte trong file. Ta định nghĩa lớp **`FileByteArray`** có chức năng như trên với chỉ mục cho phép truy cập tới byte thứ **index**. Lớp **`ReverseApp`** sẽ sử dụng một đối tượng thuộc lớp **`FileByteArray`** để đảo ngược một file.

```
using System;
using System.IO;

// Lop cho phep truy cap den file nhu la mot mang cac bye
public class FileByteArray {
     Stream st; //stream dung de truy cap den file

     //Phuong thuc khoi tao: mo fileName
     // va lien ket luong stream toi file
     public FileByteArray(string fileName)    {
          st = new FileStream(fileName, FileMode .Open);
     }

     // Dong luong (stream) sau khi da thao tac xong
     public void Close() {
          st.Close();
          st = null;
     }

     // Indexer de doc/ghi byte thu index cua file
     public byte this[long index]        {
          // Doc mot byte tai vi tri index
          get  {
               //Tao vung dem de doc ghi
               byte[] buffer = new byte[1];

               //Dua con tro den vi tri index trong file
               st.Seek(index, SeekOrigin.Begin);

               //Doc 1 byte, tai vi tri cua con tro doc
               // (offset = 0, count = 1)
               st.Read(buffer, 0, 1);
               return buffer[0];
          }

          // Ghi mot byte và file tai vi tri index .
          set  {
               //Gan gia tri can ghi vao buffer
```

```
                byte[] buffer = new byte[1] { value      };

                //Dich  chuyen  den  vi  tri  can  ghi  trong
            luong
                st.Seek(index, SeekOrigin.Begin);

                //Ghi du lieu vao file
                st.Write(buffer, 0, 1);
            }
        }

        // Thuộc tính lay chieu dai cua file (so byte)
        public long Length  {
            get  {
                //Ham seek tra ve vị trí của con trỏ.
                return st.Seek(0, SeekOrigin.End);
            }
        }
}

public class ReverseApp  {
        public static void Main()      {
            FileByteArray file;
            file = new FileByteArray("F:\\data.txt");
            long len = file.Length;
            long i;
            // dao nguoc file
            for (i = 0; i < len / 2; ++i) {
                byte t;
                t = file[i];
                file[i] = file[len - i - 1];
                file[len - i - 1] = t;
            }
            //Xuat file
            for (i = 0; i < len; ++i) {
                Console.Write((char)file[i]);
            }
            file.Close();
            Console.ReadLine();
        }
}
```

Trong lớp **`FileByteArray`** ở trên, ***chỉ mục*** trả về kiểu ***byte*** và có một tham số ***index*** là vị trí cần truy cập trong file (kiểu long).

```
public byte this[long index]
```

Phương thức ***get*** của chỉ mục này định nghĩa các dòng lệnh để đọc một byte từ file, phương thức ***set*** định nghĩa các dòng lệnh để ghi một byte vào file.

Chú ý:

- Một chỉ mục phải có ít nhất một tham số, và tham số có thể có kiểu bất kỳ.
- chỉ mục có thể chỉ có phương thức ***get***.
- Mặc dù chỉ mục là một đặc điểm thú vị của **C#** nhưng cần phải sử dụng đúng mục đích (sử dụng để đối tượng có thể hoạt động như mảng hay mảng nhiều chiều).
- Một lớp có thể có nhiều ***chỉ mục*** nhưng chúng phải có các dấu hiệu phân biệt với nhau (tương tự như định nghĩa chồng phương thức).

I.5. Lớp lồng nhau

Lớp định nghĩa bên trong một lớp khác gọi là *inner class*, lớp nằm ngoài gọi là *outer class*.

Các phương thức của lớp nằm trong có thể truy cập đến các thành phần private của lớp nằm ngoài (nhưng phải thông qua một đối tượng nào đó).

I.6. Câu hỏi ôn tập

1. *Cú pháp khai báo dữ liệu của lớp?*
2. *Sự khác nhau giữa thành viên được khai báo là public và các thành viên không được khai báo là public?*
3. *Lớp mà chúng ta xây dựng thuộc kiểu dữ liệu tham trị hay tham chiếu?*
4. *Có phải tất cả những dữ liệu thành viên luôn luôn được khai báo là public để bên ngoài có thể truy cập chúng?*
5. *Có thể tạo phương thức bên ngoài của lớp hay không?*
6. *Sự khác nhau giữa một lớp và một đối tượng của lớp?*
7. *Thành viên nào trong một lớp có thể được truy cập mà không phải tạo thể hiện của lớp?*
8. *Khi nào thì phương thức khởi tạo được gọi?*
9. *Khi nào thì phương thức khởi tạo tĩnh được gọi?*
10. *Phương thức tĩnh có thể truy cập trực tiếp thành viên nào và không thể truy cập trực tiếp thành viên nào trong một lớp?*
11. *Có thể truyền biến chưa khởi tạo vào một hàm được không?*
12. *Sự khác nhau giữa truyền biến tham chiếu và truyền biến tham trị vào một phương thức?*
13. *Làm sao để truyền tham chiếu với biến kiểu giá trị vào trong một phương thức?*
14. *Từ khóa **this** được dùng làm gì trong một lớp?*
15. *Cú pháp định nghĩa một thuộc tính?*
16. *Cú pháp định nghĩa một toán tử?*

I.7. Bài tập tổng hợp

1. *Xây dựng một lớp đường tròn lưu giữ bán kính và tâm của đường tròn. Tạo các phương thức để tính chu vi, diện tích của đường tròn.*
2. *Thêm thuộc tính BanKinh vào lớp được tạo ra từ bài tập 1.*
3. *Viết lớp giải phương trình bậc hai. Lớp này có các thuộc tính a, b, c và nghiệm x1, x2. Lớp cho phép bên ngoài xem được các nghiệm của phương trình và cho phép thiết lập hay xem các giá trị a, b, c.*
4. *Xây dựng lớp đa thức với các toán tử +, -, *, / và chỉ mục để truy cập đến hệ số thứ i của đa thức.*
5. *Xây dựng lớp ma trận với các phép toán +, -, *, / và chỉ mục để truy cập đến phần tử bất kỳ của ma trận.*
6. *Xây dựng lớp NguoiThueBao (số điện thọai, họ tên), từ đó xây dựng lớp DanhBa (danh bạ điện thọai) với các phương thức như nhập danh bạ điện thọai, xuất danh bạ điện thọai, tìm số điện thoại theo tên (chỉ mục), tìm tên theo số điện thoại (chỉ mục).*