LAB 10. QUẢN LÝ NGOẠI LỆ VÀ CÁC LỚP THƯ VIỆN

THỜI LƯỢNG : 4 TIẾT

A. Mục tiêu

- Giúp sinh viên hiểu rõ cơ chế phát sinh, bắt và xử lý lỗi (hay ngoại lệ).
- Hướng dẫn sinh viên cách sử dụng các lớp thư viện có sẵn như List, Queue, Stack, …
- Sau khi hoàn thành bài thực hành này, sinh viên cần:
  - Nắm vững cơ chế phát sinh ra ngoại lệ, bắt và xử lý các ngoại lệ
  - Sử dụng được một số lớp trong thư viện hàm xây dựng sẵn của .Net Framework
  - Tự xây dựng được các thư viện hàm có chức năng tương tự
  - Sử dụng thuần thục vòng lặp foreach
  - Hiểu rõ cách tạo các lớp Collection thực thi interface IEnumerable và IEnumerator

B. Yêu cầu

- Sinh viên phải trả lời đầy đủ các câu hỏi (nếu có) hoặc chụp màn hình kết quả, ghi lại vào tập tin Word theo yêu cầu ghi trong phần hướng dẫn thực hành.
- Đặt tên tập tin Word theo dạng: Lab10_MSSV_HoVaTen.rar.
- Tạo thư mục, đặt tên là MSSV_Lab10 để lưu tất cả bài làm trong bài thực hành này.
- Sinh viên phải hoàn thành tối thiểu 3 bài tập bắt buộc.
- Phải tạo một dự án riêng cho mỗi phần hoặc mỗi bài tập.
- Cuối buổi thực hành, nén thư mục trên & nộp bài qua email: phucnguyen5555@gmail.com

C. Hướng dẫn thực hành

1. Bắt và xử lý các lỗi xảy ra trong chương trình

Phần này hướng dẫn cách bắt và xử lý một số ngoại lệ cơ bản như lỗi sai kiểu dữ liệu, lỗi ép kiểu, lỗi tràn số, …

Bước 1. Tạo một thư mục, đặt tên theo dạng MSSV-Lab10. Ví dụ: 1210234-Lab10.
Bước 2. Tạo dự án Console Application mới, đặt tên là Lab10_Bai01_BatCacNgoaiLe
Bước 3. Tạo một lớp mới, đặt tên là ViDuNgoaiLe.
Bước 4. Cài đặt một phương thức để bắt lỗi nhập dữ liệu sai như sau

```
// Hàm này yêu cầu người dùng nhập một số
// nguyên. Nếu nhập đúng thì trả về số đó
// Nếu nhập sai thì trả về 0.
public int BatLoiKieuDuLieu()
{
    try
    {
        Console.Write("Nhap mot so nguyen : ");
        int giaTri = int.Parse(Console.ReadLine());
        return giaTri;
    }
    // Bắt tất cả các ngoại lệ có thể xảy ra
    // Phần (Exception) có thể bỏ đi
    catch (Exception)
    {
        return 0;
    }
}
```

Bước 5. Trong hàm Main, nhập đoạn mã sau để kiểm tra

```
// Tạo một thể hiện của lớp ví dụ
ViDuNgoaiLe viDu = new ViDuNgoaiLe();

// Bắt ngoại lệ nhập sai kiểu dữ liệu
int k = viDu.BatLoiKieuDuLieu();

// Xuất giá trị vừa nhập
Console.WriteLine("Gia tri vua nhap : {0}", k);
```

Bước 6. Chạy chương trình 2 lần và ghi nhận kết quả. Lần đầu, nhập một số nguyên bất kỳ. Lần thứ 2, nhập chữ cái hoặc không nhập gì cả rồi nhấn phím Enter.

Bước 7. Quay trở lại lớp ViDuNgoaiLe, cài đặt thêm phương thức sau:

```
// Ví dụ bắt lỗi và xuất thông báo lỗi
public int BatLoiVaXuatLoi()
{
    try
    {
        Console.Write("Nhap mot so nguyen : ");
        int giaTri = int.Parse(Console.ReadLine());
        return giaTri;
    }
    // Bắt tất cả các ngoại lệ có thể xảy ra
    // Phần (Exception) bắt buộc phải có
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return 0;
    }
}
```

Bước 8. Trong hàm Main, bổ sung đoạn mã sau:

```
// Bắt ngoại lệ và xuất thông báo nếu có
k = viDu.BatLoiVaXuatLoi();

// Xuất giá trị vừa nhập
Console.WriteLine("Gia tri vua nhap : {0}", k);
```

Bước 9. Chạy chương trình 2 lần (như bước 6) và ghi nhận kết quả.

Bước 10. Tiếp tục định nghĩa thêm phương thức sau ở lớp ViDuNgoaiLe

```
// Ví dụ chỉ rõ loại lỗi muốn bắt
public int ChiRoLoiMuonBat()
{
    try
    {
        Console.Write("Nhap mot so nguyen : ");
        int giaTri = int.Parse(Console.ReadLine());
        return giaTri;
    }
    // Bắt ngoại lệ liên quan tới định dạng
    // Phần (FormatException ex) bắt buộc phải có
    catch (FormatException ex)
    {
        Console.WriteLine(ex.Message);
        return 0;
    }
}
```

Bước 11. Bổ sung đoạn mã sau vào hàm Main

```
// Bắt một dạng ngoại lệ cụ thể
k = viDu.ChiRoLoiMuonBat();

// Xuất giá trị vừa nhập
Console.WriteLine("Gia tri vua nhap : {0}", k);
```

Bước 12. Chạy chương trình 2 lần (như bước 6) và ghi nhận kết quả

Bước 13. Cài đặt thêm phương thức sau vào lớp ViDuNgoaiLe

```
// Ví dụ bắt nhiều loại ngoại lệ khác nhau
public int BatNhieuLoaiNgoaiLeKhacNhau()
{
    try
    {
        Console.Write("Nhap mot so nguyen : ");
        int giaTri = int.Parse(Console.ReadLine());
        return giaTri;
    }
    // Bắt ngoại lệ liên quan tới định dạng
```

```
            // Phần (FormatException ex) bắt buộc phải có
            catch (FormatException ex)
            {
                Console.WriteLine(ex.Message);
                return 0;
            }
            // Bắt ngoại lệ tràn số khi nhập số quá lớn
            // Phần (FormatException ex) bắt buộc phải có
            catch (OverflowException ex)
            {
                Console.WriteLine(ex.Message);
                return 1;
            }
            // Khối lệnh bắt tất cả ngoại lệ phải đặt sau cùng
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                return -1;
            }
        }
```

Bước 14. Thêm đoạn mã sau vào hàm Main

```
            // Bắt nhiều loại ngoại lệ khác nhau
            k = viDu.BatNhieuLoaiNgoaiLeKhacNhau();

            // Xuất giá trị vừa nhập
            Console.WriteLine("Gia tri vua nhap : {0}", k);
```

Bước 15. Chạy chương trình 2 lần (như bước 6) và ghi nhận kết quả

Bước 16. Trong lớp ViDuNgoaiLe, bổ sung thêm phương thức sau

```
// Sử dụng đầy đủ try-catch-finally
public int SuDungKhoiFinally()
{
    try
    {
        Console.Write("Nhap mot so nguyen : ");
        int giaTri = int.Parse(Console.ReadLine());
        return giaTri;
    }
    // Bắt tất cả các ngoại lệ có thể xảy ra
    catch (Exception)
    {
        return 0;
    }
    finally
    {
        Console.WriteLine("Lenh nay thuc hien truoc khi return");
    }
}
```

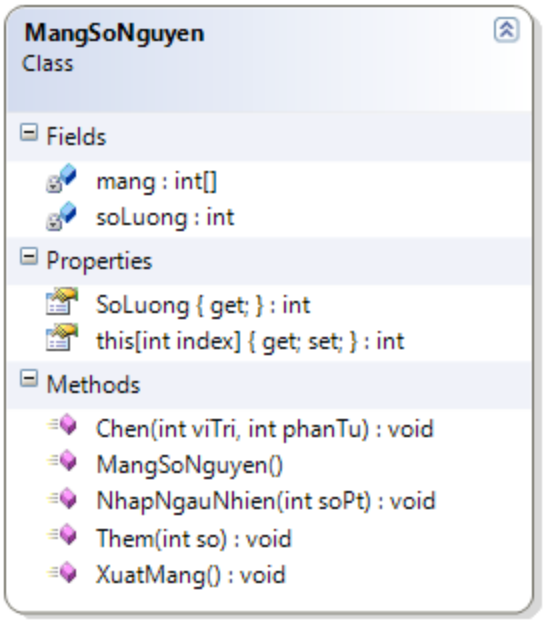
Bước 17. Thêm đoạn mã sau vào hàm Main

```
// Sử dụng thêm khối lệnh finally
k = viDu.SuDungKhoiFinally();

// Xuất giá trị vừa nhập
Console.WriteLine("Gia tri vua nhap : {0}", k);
```

Bước 18. Chạy chương trình 2 lần (như bước 6) và ghi nhận kết quả

2. Phát sinh ngoại lệ



Phần này hướng dẫn cách phát sinh ra ngoại lệ khi dữ liệu không hợp lệ hoặc có thao tác không phù hợp. Chẳng hạn như gán giá trị cho phần tử của mảng có chỉ số âm.

Bước 1. Tạo dự án Console Application mới, đặt tên là Lab10_Bai02_PhatSinhNgoaiLe

Bước 2. Tạo một lớp mới, đặt tên là MangSoNguyen (mảng số nguyên).

Bước 3. Cài đặt lớp MangSoNguyen theo hình bên hoặc sao chép từ các bài Lab trước sang.

Bước 4. Trong phần định nghĩa chỉ mục, thay đổi lại mã nguồn như sau:

```
// Định nghĩa chỉ mục
public int this[int index]
{
    get
    {
        // Nếu chỉ số không hợp lệ thì phát sinh lỗi
        if (index < 0 || index >= soLuong)
            throw new IndexOutOfRangeException("Chi so khong hop le");

        return mang[index];
    }
    set
    {
        // Nếu chỉ số không hợp lệ thì phát sinh lỗi
        if (index < 0 || index >= soLuong)
            throw new IndexOutOfRangeException("Chi so khong hop le");
        mang[index] = value;
    }
}
```

Bước 5. Định nghĩa lại phương thức Chen như sau

```
// Định nghĩa hàm chèn một phần tử mới vào vị trí
public void Chen(int viTri, int phanTu)
{
    // Kiểm tra vị trí chèn, nếu vị trí không hợp
    // lệ (số âm hoặc lớn hơn số lượng phần tử)
    // thì phát sinh ra ngoại lệ (lỗi)
    if (viTri < 0 || viTri > soLuong)
        throw new IndexOutOfRangeException();

    if (viTri == soLuong)
        Them(phanTu);
    else
    {
        // Ngược lại, vị trí hợp lệ thì chèn thêm
        for (int i = soLuong; i > viTri; i--)
            mang[i] = mang[i - 1];

        mang[viTri] = phanTu;
        soLuong++;
    }
}
```

Bước 6. Trong hàm Main, nhập đoạn mã sau để kiểm tra

```
// Tạo thể hiện của MangSoNguyen
MangSoNguyen mangNguyen = new MangSoNguyen();

// Gọi hàm nhập ngẫu nhiên
mangNguyen.NhapNgauNhien(100);

// Xuất mảng
mangNguyen.XuatMang();

// Cố tình truy xuất tới phần tử thứ -1
try
{
    Console.WriteLine(mangNguyen[-1]);
}
// Bắt lỗi vị trí không hợp lệ
catch (IndexOutOfRangeException ex)
{
    Console.WriteLine(ex.Message);
}

Console.WriteLine("".PadRight(80, '='));
```

Bước 7. Chạy chương trình và cho biết kết quả.

Bước 8. Giải thích kết quả xuất ra trên màn hình.

3. Tạo một kiểu ngoại lệ mới

Phần này hướng dẫn cách tạo ra một lớp mới, biểu diễn cho một kiểu ngoại lệ chưa được xây dựng sẵn hoặc bạn muốn tùy biến một kiểu có sẵn. Lớp mới này bắt buộc phải kế thừa từ lớp Exception.

Bước 9. Tiếp tục với dự án Lab10_Bai02_PhatSinhNgoaiLe. Tạo một lớp mới, đặt tên là LoiMienGiaTri. Cài đặt lớp này như sau:

```
// Định nghĩa một kiểu ngoại lệ mới. Khi thêm một
// phần tử có giá trị nằm ngoài miền (phạm vi)
// cho phép thì sẽ phát sinh ra ngoại lệ này
public class LoiMienGiaTri : Exception
{
    // Phương thức khởi tạo này gán sẵn
    // thông báo lỗi: Gia tri khong hop le
    public LoiMienGiaTri()
        : base("Gia tri khong hop le")
    {
    }

    // Phương thức khởi tạo này cho phép
    // tùy biến, đưa vào một thông báo lỗi khác
    public LoiMienGiaTri(string message)
        : base(message)
    {
    }
}
```

Bước 10. Sửa đổi lại phương thức Them của lớp MangSoNguyen như sau

```
// Định nghĩa hàm thêm một số nguyên vào cuối mảng
public void Them(int so)
{
    // Kiểm tra miền giá trị của số
    if (so < -1000 || so > 1000)
        throw new LoiMienGiaTri("Gia tri hop le phai tu -1000 den 1000");

    mang[soLuong] = so;
    soLuong++;
}
```

Bước 11. Sửa lại bộ truy xuất **set** trong chỉ mục (bước 4) ở lớp MangSoNguyen

```
// Nếu chỉ số không hợp lệ thì phát sinh lỗi
if (index < 0 || index >= soLuong)
    throw new IndexOutOfRangeException("Chi so khong hop le");

// Kiểm tra miền giá trị của số
if (value < -1000 || value > 1000)
    throw new LoiMienGiaTri();

mang[index] = value;
```

Bước 12. Cập nhật lại phương thức Chen như sau

```
// Định nghĩa hàm chèn một phần tử mới vào vị trí
public void Chen(int viTri, int phanTu)
{
    // Kiểm tra vị trí chèn, nếu vị trí không hợp
    // lệ (số âm hoặc lớn hơn số lượng phần tử)
    // thì phát sinh ra ngoại lệ (lỗi)
    if (viTri < 0 || viTri > soLuong)
        throw new IndexOutOfRangeException();

    if (viTri == soLuong)
        Them(phanTu);
    else
    {
        // Kiểm tra miền giá trị của số
        if (phanTu < -1000 || phanTu > 1000)
            throw new LoiMienGiaTri();

        // Ngược lại, vị trí hợp lệ thì chèn thêm
        for (int i = soLuong; i > viTri; i--)
            mang[i] = mang[i - 1];

        mang[viTri] = phanTu;
        soLuong++;
    }
}
```

Bước 13. Chạy lại chương trình và ghi nhận kết quả. So sánh với kết quả ở bước 7.

Bước 14. Chạy lại chương trình, khi nhập giá trị mới để chèn vào mảng, hãy nhập một giá trị lớn hơn 1000 và cho biết kết quả xuất ra màn hình.

Bước 15. Trong hàm Main, bổ sung thêm đoạn mã sau

```
// Giả sử miền giá trị quy định là [-1000..1000]
// Cố tình thêm một phần tử mới ngoài miền này
try
{
    mangNguyen.Them(2567);
}
// Bắt lỗi vượt khỏi miền giá trị
catch (LoiMienGiaTri ex)
{
    Console.WriteLine("Khong the them phan tu moi");
    Console.WriteLine(ex.Message);
}
```

Bước 16. Chạy lại chương trình và ghi nhận kết quả.

Bước 17. Nếu trong đoạn mã trên, LoiMienGiaTri trong mệnh đề catch được thay bằng Exception thì kết quả xuất ra là gì?

4. Sử dụng một số lớp Collection đã được xây dựng sẵn

Phần này và các phần tiếp theo sẽ hướng dẫn cách sử dụng một số lớp đã được xây dựng sẵn trong .Net Framework để biểu diễn một danh sách các đối tượng. Các lớp này bao gồm: ArrayList, List<T> (kiểu mảng & danh sách), Dictionary<K,V> (kiểu từ điển), Queue<T> (kiểu hàng đợi), Stack<T> (kiểu ngăn xếp). Mặc dù chúng đều chứa danh sách các đối tượng nhưng cách thức truy xuất tới các phần tử rất khác nhau.

Bước 1. Tạo dự án Console Application mới,. đặt tên là Lab10_Bai03_CacLopCollection

Bước 2. Tạo lớp mới, đặt tên là ViDuVeArrayList. Bổ sung lệnh using System.Collection;

Bước 3. Định nghĩa lớp này như sau:

```
class ViDuVeArrayList
{
    // Ví dụ cách tạo một đối tượng ArrayList
    public void TaoMang()
    {
        // Khởi tạo mảng với số phần tử = 4
        ArrayList danhSach1 = new ArrayList();

        // Khởi tạo mảng với số phần tử = 100
        ArrayList danhSach2 = new ArrayList(100);

        object[] cacPt = new object[]
        {
            1, "abc", 2.25, new Random()
        };

        // Khởi tạo mảng với các phần tử cho trước
        ArrayList danhSach3 = new ArrayList(cacPt);
    }

    // Xuất các phần tử của mảng
    public void XuatMangDungFor(ArrayList danhSach)
    {
        for (int i = 0; i < danhSach.Count; i++)
        {
            Console.Write("{0}\t", danhSach[i]);
        }
    }

    // Xuất các phần tử của mảng
    // Lớp ArrayList thực thi interface IEnumerator nên
    // có thể dùng lệnh foreach để duyệt qua từng phần tử
    public void XuatMangDungForEach(ArrayList danhSach)
    {
        foreach (object phanTu in danhSach)
        {
            Console.Write("{0}\t", phanTu);
        }
    }
```

```
// Xuất các phần tử của mảng
// Lớp ArrayList thực thi interface IEnumerator nên
// có thể dùng lệnh foreach để duyệt qua từng phần tử
public void XuatMangDungForEach(ArrayList danhSach)
{
    foreach (object phanTu in danhSach)
    {
        Console.Write("{0}\t", phanTu);
    }
}

// Ví dụ cách thêm hoặc chèn thêm phần tử
public void CacHamChenThem()
{
    // Khởi tạo mảng với số phần tử = 4
    ArrayList danhSach = new ArrayList();

    // Thêm một phần tử vào cuối danhSach
    danhSach.Add('@');
    danhSach.Add(15.75f);

    // Thêm nhiều phần tử vào cuối danhSach
    object[] cacPt = new object[]
    {
        1, "abc", 2.25, '$'
    };
    danhSach.AddRange(cacPt);

    // Chèn thêm phần tử vào vị trí
    danhSach.Insert(1, 40);
    danhSach.Insert(0, "Blah");

    XuatMangDungFor(danhSach);
}

// Ví dụ cách xóa các phần tử
public void CacHamXoa()
{
    Random rd = new Random();
    ArrayList danhSach = new ArrayList();

    // Thêm ngẫu nhiên 20 số vào danh sách
    for (int i = 0; i < 20; i++)
    {
        danhSach.Add(rd.Next(100));
    }
    XuatMangDungFor(danhSach);

    // Xóa phần tử 100
    danhSach.Remove(100);

    // Xóa phần tử thứ 10
    danhSach.RemoveAt(10);
```

```
            // xóa 12 phần tử liên tiếp kể tự vị trí 5
            danhSach.RemoveRange(5, 12);

            XuatMangDungFor(danhSach);
        }

        // Ví dụ cách truy cập tới các phần tử
        public void TruyCapPhanTu()
        {
            Random rd = new Random();
            ArrayList danhSach = new ArrayList();

            // Thêm ngẫu nhiên 20 số vào danh sách
            for (int i = 0; i < 20; i++)
            {
                danhSach.Add(rd.Next(100));
            }
            XuatMangDungFor(danhSach);

            // Tính tổng các phần tử
            int tong = 0;
            for (int i = 0; i < 20; i++)
            {
                tong += (int)danhSach[i];
            }
            Console.WriteLine("Tong cac phan tu = {0}", tong);
        }
    }
```

Bước 4. Trong hàm Main, nhập đoạn mã sau:

```
            ViDuVeArrayList viDu = new ViDuVeArrayList();

            viDu.TruyCapPhanTu();

            viDu.CacHamXoa();

            Console.ReadKey();
```

Bước 5. Chạy chương trình và ghi nhận kết quả.

Bước 6. Giải thích một cách ngắn gọn hoạt động của chương trình vừa tạo.

5. Sử dụng lớp List<T>

Lớp List<T> dùng để biểu diễn một danh sách hay một mảng các phần tử có kiểu T. Khác biệt chủ yếu giữa ArrayList và List<T> là ở chổ mỗi phần tử của List<T> có kiểu là T trong khi ArrayList có kiểu là object. Vì thế, muốn lấy giá trị phần tử trong ArrayList, ta phải ép kiểu.

Bước 7. Tiếp tục với dự án Lab10_Bai03. Tạo và định nghĩa lớp ViDuVeLopList như sau

```
class ViDuVeLopList
{
    // Ví dụ cách tạo một đối tượng ArrayList
    public void TaoMang()
    {
        // Khởi tạo mảng với số phần tử = 4
        List<string> mangChuoi = new List<string>();

        // Khởi tạo mảng với số phần tử = 100
        List<int> mangNguyen = new List<int>(100);

        double[] cacPt = new double[]
        {
            1, 10.03, 2.25
        };
        // Khởi tạo mảng với các phần tử cho trước
        List<double> mangThuc = new List<double>(cacPt);
    }

    // Xuất các phần tử của mảng
    public void XuatMangDungFor(List<int> danhSach)
    {
        for (int i = 0; i < danhSach.Count; i++)
        {
            Console.Write("{0}\t", danhSach[i]);
        }
    }

    // Xuất các phần tử của mảng
    // Lớp List<T> cũng thực thi interface IEnumerator nên
    // có thể dùng lệnh foreach để duyệt qua từng phần tử
    public void XuatMangDungForEach(List<int> danhSach)
    {
        foreach (int phanTu in danhSach)
        {
            Console.Write("{0}\t", phanTu);
        }
    }


    // Ví dụ cách thêm hoặc chèn thêm phần tử
    public void CacHamChenThem()
    {
        // Khởi tạo mảng với số phần tử = 4
        List<int> danhSach = new List<int>();

        // Thêm một phần tử vào cuối danhSach
        danhSach.Add(20);
        danhSach.Add(15);

        // Thêm nhiều phần tử vào cuối danhSach
        int[] cacPt = new int[] { 1, 298, 25, 112 };
        danhSach.AddRange(cacPt);
```

```
        // Chèn thêm phần tử vào vị trí
        danhSach.Insert(1, 40);
        danhSach.Insert(0, 27);

        XuatMangDungFor(danhSach);
    }

    // Ví dụ cách xóa các phần tử
    public void CacHamXoa()
    {
        Random rd = new Random();
        List<int> danhSach = new List<int>();

        // Thêm ngẫu nhiên 20 số vào danh sách
        for (int i = 0; i < 20; i++)
        {
            danhSach.Add(rd.Next(100));
        }
        XuatMangDungFor(danhSach);

        // Xóa phần tử 100
        danhSach.Remove(100);

        // Xóa phần tử thứ 10
        danhSach.RemoveAt(10);

        // xóa 12 phần tử liên tiếp kể tự vị trí 5
        danhSach.RemoveRange(5, 12);

        XuatMangDungFor(danhSach);
    }

    // Ví dụ cách truy cập tới các phần tử
    public void TruyCapPhanTu()
    {
        Random rd = new Random();
        List<int> danhSach = new List<int>();

        // Thêm ngẫu nhiên 20 số vào danh sách
        for (int i = 0; i < 20; i++)
            danhSach.Add(rd.Next(100));
        XuatMangDungFor(danhSach);

        // Tính tổng các phần tử
        int tong = 0;
        for (int i = 0; i < 20; i++)
            tong += danhSach[i];

        Console.WriteLine("Tong cac phan tu = {0}", tong);
    }
}
```

Bước 8. Trong hàm Main, thay đổi đoạn mã trong bước 3 thành đoạn mã sau:

```
ViDuVeLopList viDu = new ViDuVeLopList();

viDu.TruyCapPhanTu();

viDu.CacHamXoa();

Console.ReadKey();
```

Bước 9. Chạy chương trình & ghi lại kết quả. Mô tả vắn tắt cách hoạt động của chương trình.

6. Lớp Stack<T>

Stack<T> dùng để lưu trữ một danh sách các phần tử có kiểu T. Trong đó, phần tử được đưa vào sau sẽ được lấy ra trước (LIFO – Last In First Out). Phần này hướng dẫn cách áp dụng lớp Stack<T> vào bài toán đổi một số từ 10 sang một hệ khác.

Bước 10. Tiếp tục dự án Lab10_Bai03. Tạo và định nghĩa lớp ViDuVeLopStack như sau

```
class ViDuVeLopStack
{
    // Định nghĩa phương thức đổi một số (so) từ
    // cơ số 10 sang cơ số khác (coSo)
    public string DoiCoSo(uint so, uint coSo)
    {
        // Khởi tạo ngăn xếp để chứa các số dư
        Stack<uint> cacSoDu = new Stack<uint>();

        while (so > 0)
        {
            // Đưa số dư vào ngăn xếp
            cacSoDu.Push(so % coSo);

            // Tính thương số của so / coSo
            so /= coSo;
        }
        // Trả về chuỗi biểu diễn số ở cơ số mới
        return GhepSo(cacSoDu);
    }

    // Định nghĩa phương thức để ghép các số dư
    // trong Stack để tạo ra số ở cơ số mới
    private string GhepSo(Stack<uint> nganXep)
    {
        // Tạo đối tượng để nối chuỗi
        StringBuilder sb = new StringBuilder();

        // Chừng nào ngăn xếp còn chứa phần tử
        while (nganXep.Count > 0)
```

```
        {
            // thì lấy phần tử đó ra bằng hàm Pop
            uint soDu = nganXep.Pop();

            // Kiểm tra số dư, nếu nhỏ hơn 10
            // (0..9) thì nối số đó vào chuỗi
            if (soDu < 10)
                sb.Append(soDu);
            else
                // Nếu lớn hơn hoặc bằng 10 thì
                // chuyển sang chữ cái.
                sb.Append((char) (soDu + 55));
        }
        return sb.ToString();
    }
}
```

Bước 11. Trong hàm Main, thay đoạn mã trong bước 8 thành đoạn mã sau

```
ViDuVeLopStack vdNganXep = new ViDuVeLopStack();

uint so = 1000, coSo = 16;

// Đổi số 1000 từ hệ 10 sang hệ 16
string ketQua = vdNganXep.DoiCoSo(so, coSo);

// Xuất kết quả
Console.WriteLine("{0}(10) = {1}({2})",
                  so, coSo, ketQua);
```

Bước 12. Chạy chương trình và cho biết kết quả.

Bước 13. Hãy thay đổi giá trị của biến so và coSo rồi chạy lại chương trình để kiểm tra kết quả

Bước 14. Chương trình trên chỉ hoạt động đúng khi đổi sang những cơ số nào?

7. Lớp Queue<T>

Queue<T> dùng để chứa một danh sách các phần tử có kiểu T. Trong đó, những phần tử được đưa vào trước sẽ được lấy ra trước. Phần này hướng dẫn cách áp dụng lớp Queue<T> vào bài toán đảo ngược một số.

Bước 15. Tiếp tục dự án Lab10_Bai03. Tạo và định nghĩa lớp ViDuVeLopQueue như sau

```
class ViDuVeLopQueue
{
    // Định nghĩa phương thức đảo ngược 1 số
    public uint DaoNguocSo(uint so)
    {
        // Khởi tạo hàng đợi để chứa các số
        Queue<uint> cacChuSo = new Queue<uint>();
```

```
            // Lần lượt lấy từng chữ số, bắt đầu
            // từ hàng đơn vị (từ phải -> trái)
            while (so > 0)
            {
                // Đưa số dư vào ngăn xếp
                cacChuSo.Enqueue(so % 10);

                // Tính thương số của so / coSo
                so /= 10;
            }
            // Trả về chuỗi biểu diễn số ở cơ số mới
            return GhepSo(cacChuSo);
        }

        // Định nghĩa phương thức để ghép các số
        // trong Queue để tạo ra một số mới
        private uint GhepSo(Queue<uint> hangDoi)
        {
            uint soMoi = 0;

            // Chừng nào hàng đợi còn chứa phần tử
            while (hangDoi.Count > 0)
            {
                // thì lấy phần tử đó ra bằng hàm Dequeue
                uint soDu = hangDoi.Dequeue();

                // Tạo dần số mới
                soMoi = soMoi * 10 + soDu;
            }
            return soMoi;
        }
    }
```

Bước 16. Thay đổi đoạn mã trong hàm Main bởi đoạn mã sau

```
ViDuVeLopQueue vdHangDoi = new ViDuVeLopQueue();
uint so = 1234500;

// Lấy số lượng chữ số của so
int len = so.ToString().Length;

// Đảo ngược chữ số
uint daoSo = vdHangDoi.DaoNguocSo(so);

// Đổi thành chuỗi có cùng số chữ số với so
string chuoiSo = daoSo.ToString().PadLeft(len, '0');

// Xuất kết quả
Console.WriteLine("{0} => {1} => {2}",
                  so, daoSo, chuoiSo);
```

Bước 17. Chạy chương trình và ghi nhận kết quả.

8. Lớp Dictionary<K, V>

Dictionary<K, V> chứa một tập các phần tử. Trong đó, mỗi phần tử là một cặp hai thành phần: Key (có kiểu K) và Value (có kiểu V). K và V có thể là các kiểu bất kỳ. Cặp key/value như vậy được biểu diễn bởi lớp KeyValuePair<K, V>.

Phần này hướng dẫn cách áp dụng lớp Dictionary<K, V> vào việc thống kê số lần xuất xuất của các chữ cái, chữ số, ký tự đặc biệt trong một câu.

Bước 18. Tiếp tục dự án Lab10_Bai03. Tạo và định nghĩa lớp ViDuVeDictionary như sau

```
class ViDuVeDictionary
{
    // Định nghĩa phương thức thống kê số lần xuất hiện của các
    // chữ cái, các chữ số hay các ký tự đặc biệt trong câu.
    public void ThongKeKyTu(string cau)
    {
        // Khởi tạo Dictionary. Key có kiểu char, Value kiểu int
        Dictionary<char, int> duLieuTk = new Dictionary<char, int>();

        // duyệt qua từng ký tự trong câu
        foreach (char kyTu in cau)
        {
            // Nếu ký tự đó đã được thống kê trước đó
            if (duLieuTk.ContainsKey(kyTu))
                duLieuTk[kyTu]++;       // Thì tăng số lần xuất hiện
            else
                // Nếu chưa thì thêm ký tự và số lần xuất hiện là 1
                duLieuTk.Add(kyTu, 1);
        }
        // Xuất các ký tự và số lần xuất hiện tương ứng
        XuatKetQua(duLieuTk);
    }

    // Định nghĩa phương thức xuất các giá trị thống kê
    private void XuatKetQua(Dictionary<char, int> ketQuaTk)
    {
        // Duyệt qua từng cặp giá trị key/value trong Dictionary
        foreach (KeyValuePair<char, int> cap in ketQuaTk)
        {
            Console.WriteLine("{0} : {1}", cap.Key, cap.Value);
        }
    }
}
```

Bước 19. Trong hàm Main, thay thế đoạn mã trong bước 16 bởi đoạn mã sau

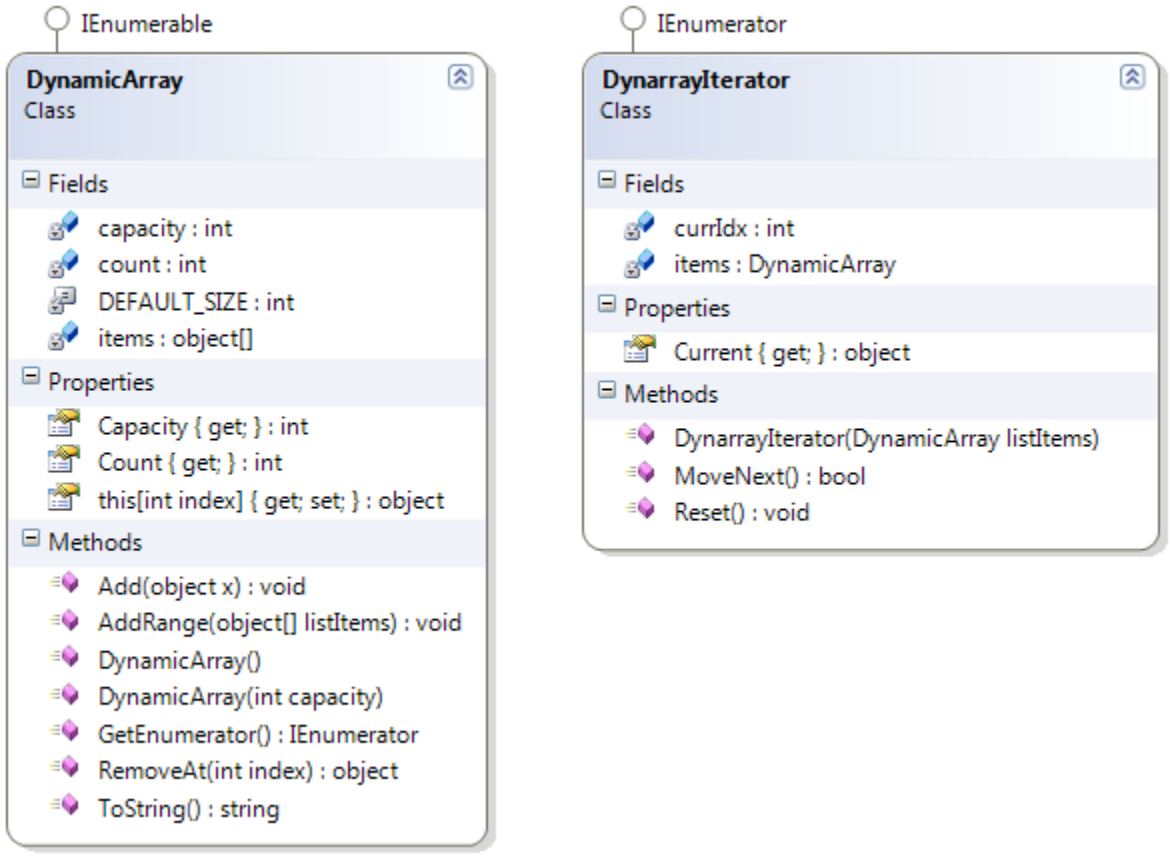
```
ViDuVeDictionary vdTuDien = new ViDuVeDictionary();
string cau = "Kho@ C0ng nghE Thong TiN - Truong d@i hoc Da Lat";
vdTuDien.ThongKeKyTu(cau);
```

Bước 20. Chạy chương trình và ghi nhận kết quả.

9. Xây dựng lớp mảng động

Trong các phần trên, câu lệnh foreach được dùng để duyệt lần lượt qua từng phần tử của mảng. Câu lệnh này được áp dụng cho cách kiểu mảng, ArrayList, List, Dictionary, … Trong các Lab trước, ta đã định nghĩa nhiều lớp biểu diễn mảng các đối tượng như số nguyên, hình học, nhân viên, sách, … Tuy nhiên, không thể sử dụng lệnh foreach đối với các đối tượng kiểu danh sách này. Lý do là lớp danh sách ta đã tạo không thực thi interface IEnumerable. Ngoài ra, số lượng phần tử có thể lưu trong các danh sách đó là có giới hạn.

Trong phần này, ta sẽ tạo một lớp DynamicArray (mảng động) để lưu trữ một danh sách các đối tượng bất kỳ (mỗi phần tử có kiểu là object). Khi số lượng phần tử vượt quá khả năng lưu trữ cho phép, mảng sẽ tự động tăng kích thước. Mặt khác, lớp này sẽ thực thi interface IEnumerable để cho phép sử dụng được vòng lặp foreach.



Sơ đồ các lớp sẽ cài đặt

Bước 1. Tạo dự án Console Application mới, đặt tên là Lab10_Bai04_MangDong

Bước 2. Tạo hai lớp mới, đặt tên lần lượt là: DynamicArray, DynarrayIterator

Bước 3. Cài đặt lớp DynamicArray như sau:

```
// Định nghĩa lớp DynamicArray để lưu trữ một
// danh sách các phần tử có kiểu bất kỳ.
// Kích thước của danh sách tự thay đổi.
public class DynamicArray
{
    // Hằng số cho biết dung lượng mặc định là 16 phần tử
    private const int DEFAULT_SIZE = 16;

    // Khai báo mảng lưu các đối tượng
    private object[] items;

    // Khai báo biến đếm số lượng phần tử
    // đã được lưu trong mảng
    private int count;

    // Khai báo biến lưu dung lượng của mảng
    private int capacity;

    // Định nghĩa thuộc tính lấy số lượng phần
    // tử đã được lưu trong mảng
    public int Count
    {
        get { return count; }
    }

    // Thuộc tính lấy số lượng phần tử tối đa có thể lưu
    public int Capacity
    {
        get { return capacity; }
    }

    // Định nghĩa chỉ mục để lấy hoặc gán
    // phần tử thông qua chỉ mục.
    public object this[int index]
    {
        get
        {
            if (index < 0 || index >= count)
                throw new IndexOutOfRangeException();
            return items[index];
        }
        set
        {
            if (index < 0 || index >= count)
                throw new IndexOutOfRangeException();
            items[index] = value;
        }
    }
```

```
// Định nghĩa phương thức khởi tạo
public DynamicArray()
{
    items = new object[DEFAULT_SIZE];
    capacity = DEFAULT_SIZE;
    count = 0;
}

// Capacity là số lượng phần tử tối đa có thể lưu
public DynamicArray(int capacity)
{
    // Phát sinh lỗi nếu dung lượng nhỏ hơn 2
    if (capacity < 2)
        throw new ArgumentOutOfRangeException(
            "capacity",
            "Mảng phải chứa tối thiểu 2 phần tử");

    this.capacity = capacity;
    items = new object[capacity];
    count = 0;
}

// Định nghĩa phương thức thêm phần tử vào cuối mảng
public void Add(object x)
{
    if (x == null)
        throw new ArgumentNullException("x");

    // Nếu số phần tử đã đạt mức tối đa
    if (count == capacity)
    {
        // thì tăng dung lượng lên gấp đôi
        capacity *= 2;

        // Tạo một mảng tạm để lưu các phần tử cũ
        object[] temp = new object[capacity];

        // Sao chép các phần tử từ mảng hiện có
        // sang mảng mới với kích thước lớn hơn

        Array.Copy(items, 0, temp, 0, count);

        // Dùng mảng mới để chứa các phần tử
        items = temp;
    }

    items[count] = x;
    count++;
}

// Định nghĩa phương thức thêm nhiều phần tử
public void AddRange(object[] listItems)
{
```

```
    // Nếu tổng số phần tử vượt mức tối đa
    if (count + listItems.Length > capacity)
    {
        // thì tăng dung lượng lên gấp đôi
        capacity *= 2;

        // Tạo một mảng tạm để lưu các phần tử cũ
        object[] temp = new object[capacity];

        // Sao chép các phần tử từ mảng hiện có
        // sang mảng mới với kích thước lớn hơn
        Array.Copy(items, 0, temp, 0, count);

        // Dùng mảng mới để chứa các phần tử
        items = temp;
    }

    // Chép các phần tử mới vào mảng hiện có
    Array.Copy(listItems, 0, items, count, listItems.Length);
    count += listItems.Length;
}

// Định nghĩa hàm xóa phần tử tại vị trí
// Trả về phần tử bị xóa
public object RemoveAt(int index)
{
    // Phát sinh lỗi nếu chỉ số không hợp lệ
    if (index < 0 || index >= count)
        throw new IndexOutOfRangeException();

    // Lấy phần tử bị xóa
    object deleted = items[index];

    // xóa phần tử
    for (int i = index; i < count - 1; i++)
        items[i] = items[i + 1];

    // Giảm số phần tử đi 1
    count--;

    // Nếu số phần tử còn lại ít hơn 1 nữa dung lượng tối đa
    if (count - 1 < capacity / 2)
    {
        // thì giảm dung lượng xuống 1 nửa
        capacity /= 2;

        // Tạo một mảng tạm để lưu các phần tử cũ
        object[] temp = new object[capacity];

        // Sao chép các phần tử từ mảng hiện có
        // sang mảng mới với kích thước lớn hơn
        Array.Copy(items, 0, temp, 0, count);
```

```
            // Dùng mảng mới để chứa các phần tử
            items = temp;
        }

        return deleted;
    }

    // Chuyển tất cả các đối tượng thành chuỗi mô tả chúng
    public override string ToString()
    {
        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < count; i++)
        {
            sb.AppendLine(items[i].ToString());
        }
        return sb.ToString();
    }
}
```

Bước 4. Cài đặt lớp DynarrayIterator như sau:

```
// Lớp này dùng để duyệt lần lượt qua từng phần tử
// của danh sách bằng cách dùng hàm MoveNext và lấy
// ra phần tử thông qua thuộc tính Current.
internal class DynarrayIterator : IEnumerator
{
    // Khai báo biến lưu chỉ số của phần tử hiện tại
    private int currIdx;

    // Khai báo biến lưu danh sách các phần tử
    private DynamicArray items;

    // Định nghĩa phương thức khởi tạo
    public DynarrayIterator(DynamicArray listItems)
    {
        this.items = listItems;
        this.currIdx = -1;
    }

    // Chuyển đến phần tử tiếp theo. Trả về
    // true nếu còn phần tử sau nó.
    public bool MoveNext()
    {
        currIdx++;
        return currIdx < items.Count;
    }

    // Thiết lập lại chỉ số bắt đầu duyệt
    public void Reset()
    {
        currIdx = -1;
    }
```

```
        // Lấy phần tử hiện tại
        public object Current
        {
            get
            {
                if (currIdx < 0 || currIdx >= items.Count)
                    throw new IndexOutOfRangeException();
                return items[currIdx];
            }
        }
    }
```

Bước 5. Trở lại lớp DynamicArray, bổ sung mã để lớp này thực thi interface IEnumerable

```
public class DynamicArray : IEnumerable
```

Bước 6. Bổ sung thêm phương thức sau vào lớp DynamicArray

```
// Định nghĩa phương thức lấy đối tượng dùng
// để duyệt lần lượt từng phần tử của mảng .
public IEnumerator GetEnumerator()
{
    return new DynarrayIterator(this);
}
```

Bước 7. Trong hàm Main, nhập đoạn mã sau để kiểm tra các hàm thêm phần tử

```
DynamicArray mang = new DynamicArray();

for (int i = 0; i < 30; i++)
{
    mang.Add(i);
}
Console.WriteLine(mang);
Console.WriteLine("".PadRight(79, '='));

Console.WriteLine("Count = {0}", mang.Count);
Console.WriteLine("Capacity = {0}", mang.Capacity);
Console.WriteLine("".PadRight(79, '='));

mang.AddRange(new object[]{123, 456, 789, 890, 312});

Console.WriteLine(mang);
Console.WriteLine("Count = {0}", mang.Count);
Console.WriteLine("Capacity = {0}", mang.Capacity);
Console.WriteLine("".PadRight(79, '='));
```

Bước 8. Chạy chương trình và cho biết kết quả

Bước 9. Tiếp tục bổ sung đoạn mã dưới đây vào hàm Main để kiểm tra hàm xóa phần tử

```
mang.RemoveAt(15);
mang.RemoveAt(1);
mang.RemoveAt(29);
mang.RemoveAt(10);

Console.WriteLine(mang);
Console.WriteLine("Count = {0}", mang.Count);
Console.WriteLine("Capacity = {0}", mang.Capacity);
Console.WriteLine("".PadRight(79, '='));
```

Bước 10. Chạy chương trình và cho biết kết quả

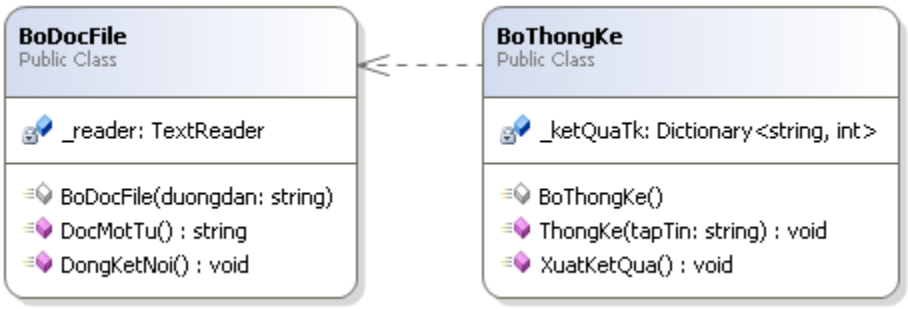
Bước 11. Nhập thêm đoạn mã sau để xuất các phần tử dùng lệnh foreach

```
// Duyệt qua từng phần tử của mảng và xuất nó
foreach (object phanTu in mang)
{
    Console.Write("{0}\t", phanTu);
}
```

Bước 12. Chạy chương trình và cho biết kết quả

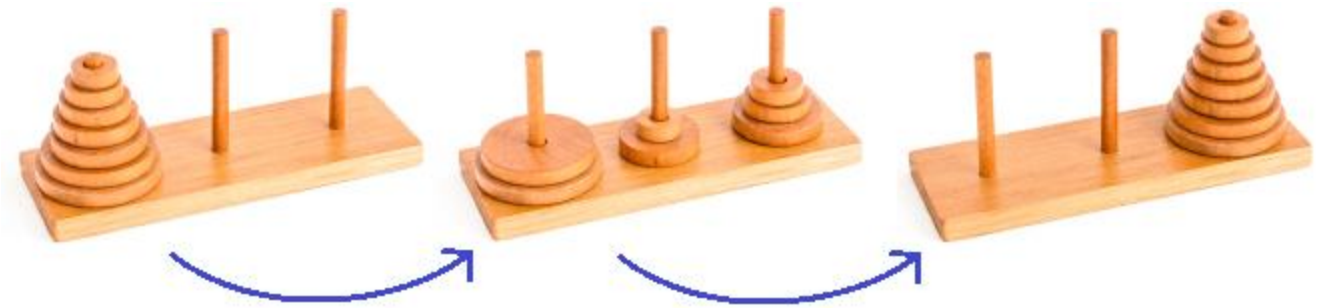
D. Bài tập bắt buộc

1. Hãy viết chương trình đọc một tập tin văn bản. Sau đó, thống kê số lần xuất hiện của mỗi từ trong văn bản đó. Chương trình cần kiểm tra tập tin có tồn tại hay không và bắt lỗi nếu không tồn tại tập tin cần đọc.



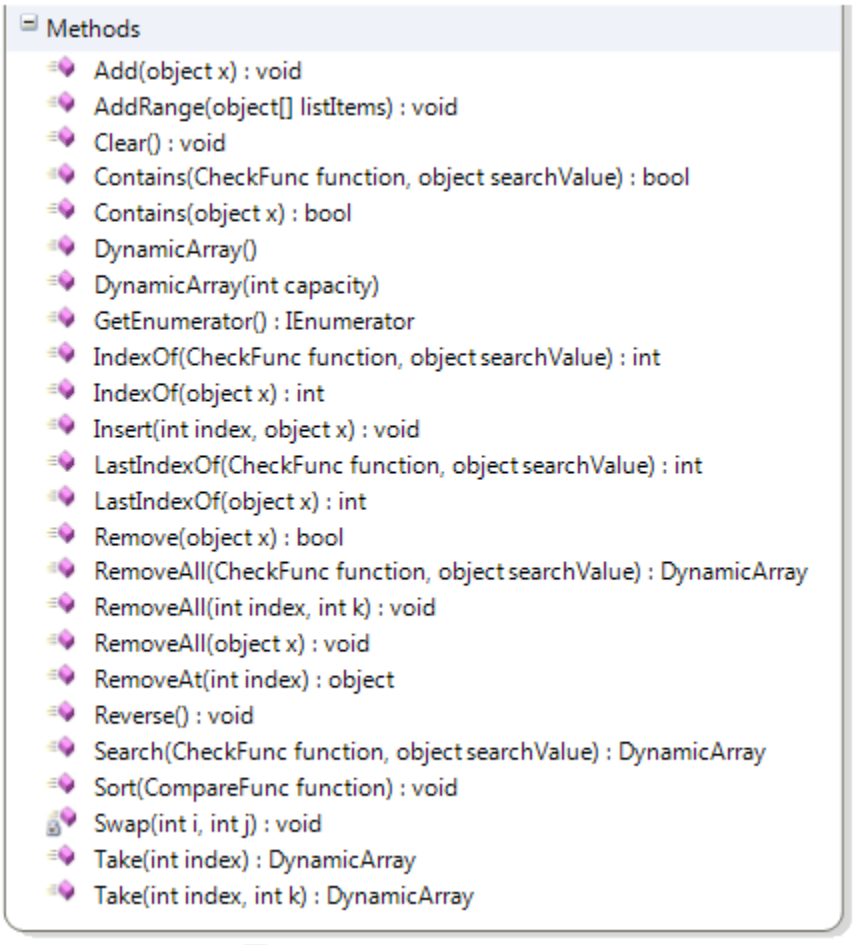
2. Áp dụng lớp Stack<T> để giải bài toán đảo ngược một chuỗi.
3. Áp dụng lớp Stack<T> để giải bài toán tháp Hà Nội. Bài toán này được phát biểu như sau:

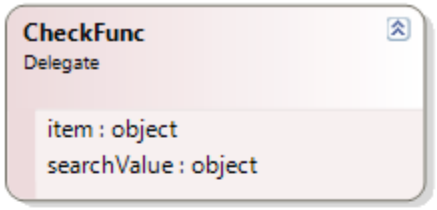
   Có 3 chiếc cọc dựng thẳng đứng và n chiếc đĩa với kích thước khác nhau. Giữa các đĩa có đục lỗ để có thể xuyên các cọc như hình bên dưới. Ban đầu, tất cả n đĩa đều được xếp ở cọc thứ nhất, đĩa nhỏ nằm trên đĩa lớn. Bài toán đặt ra là hãy chuyển tất cả n đĩa từ cọc thứ nhất sang cọc thứ ba với quy tắc: Mỗi lần chỉ chuyển một đĩa, không được đặt đĩa lớn lên trên đĩa nhỏ.

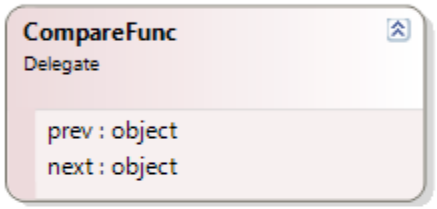


Đọc thêm: http://vnexpress.net/tin-tuc/khoa-hoc/chuyen-la/bai-toan-thap-ha-noi-2046890.html

4. Hãy cài đặt thêm các phương thức sau đây cho lớp mảng động ở phần 9.







5. Hãy cài đặt lại lớp mảng động ở phần 9 và bài tập 4 nhưng thay vì sử dụng các delegate làm điều kiện tìm kiếm và sắp xếp, ta sử dụng interface để thực hiện việc này.

   Xem lại Lab08_Bai05 (bước 9, mục 7, Lab 8) nếu cần.

6. Bổ sung các sự kiện sau vào lớp DynamicArray ở bài tập 4 và 5.

- Added: phát sinh khi thêm hay chèn một phần tử mới vào mảng thành công. Dữ liệu đi kèm theo sự kiện bao gồm: phần tử mới thêm (hay chèn) và vị trí của nó.
- Removed: phát sinh khi xóa một phần tử khỏi mảng thành công. Dữ liệu đi kèm theo sự kiện bao gồm: phần tử bị xóa và vị trí của phần tử đó.
- SizeChanged: phát sinh khi dung lượng (số phần tử tối đa có thể chứa) của mảng thay đổi. Dữ liệu đi kèm theo sự kiện là dung lượng trước và sau khi thay đổi cùng với số phần tử hiện có.

E. Bài tập làm thêm

1. Cho một bản đồ kho báu được biểu diễn bởi ma trận kích thước MxN. Giả sử một người đang đứng ở ô có tọa độ (i, j) thì có thể đi đến được 4 ô lân cận có tọa độ (i-1, j), (i+1, j), (i, j-1), (i, j+1). Mỗi phần tử của ma trận chứa số 0 hoặc 1. Số 1 nghĩa là có thể di chuyển hay đi tới được ô tương ứng, số 0 nếu ngược lại.
   a. Hãy tìm mọi đường đi có thể từ ô có tọa độ (x, y) đến ô có tọa độ (p, q).
   b. Hãy tìm một đường đi ngắn nhất từ ô có tọa độ (x, y) đến ô có tọa độ (p, q)

2. Bài toán sinh từ theo thứ tự từ điển.

   Cho n chữ cái đôi một khác nhau. Hãy tìm và xuất lần lượt tất cả những từ (có chiều dài n, không quan tâm tới nghĩa của nó) được tạo thành từ n chữ cái đó theo thứ tự từ điển. Ví dụ: Giả sử tập chữ cái là {a,b,c,d,e,f,g,h}. Nếu từ vừa mới được sinh ra là ehcgfdba thì từ tiếp theo (theo thứ tự từ điển) phải là ehdabcfg.

   Gợi ý: Bài toán này có thể được giải bằng cách áp dụng một ngăn xếp và một hàng đợi như sau:

   - Ban đầu, lần lượt đưa các chữ cái của từ mới sinh ra vào Stack, Queue đang rỗng.
   - Lấy một chữ cái từ Stack đưa vào Queue. Lặp lại việc này cho đến khi nào gặp một chữ cái đứng trước (theo bảng chữa cái ABC) chữ cái đã lấy trước đó. Gọi chữ cái này là pivot.
   - Lấy một chữ cái từ Queue rồi lại đưa vào Queue (thực chất là chuyển nó từ đầu hàng đợi tới cuối hàng đợi). Lặp lại việc này cho tới khi gặp một chữ cái đứng sau pivot (theo bảng chữ cái). Đưa chữ cái này vào Stack.
   - Đưa chữ cái từ pivot vào Queue.
   - Lấy một chữ cái từ Queue rồi lại đưa vào Queue. Lặp lại việc này cho tới khi gặp một chữ cái đứng trước pivot (theo bảng chữ cái). Đưa chữ cái này vào Stack.
   - Lần lượt lấy các chữ cái trong Queue rồi đưa vào Stack cho đến khi nào Queue rỗng thì dừng.

3. Cài đặt lớp BinaryTree (cây nhị phân) theo mô tả sau