codingOn

컬렉션

컬렉션

- 배열처럼 데이터 집합을 다루는 클래스
- 제네릭을 제외한 컬렉션은 자료형에 상관 없이 데이터를 추가하는 것이 가능

네임스페이스	컬렉션 이름
System	Array
System.Collections	Stack, Queue, ArrayList
System.Collections.Generic	List<T>, LinkedList<T>, Stack<T>, Queue<T>
System.Collections.Concurrent	ConcurrentStack<T>, ConcurrentQueue<T>

컬렉션

```
1   using System;
2   using System.Collections;
3   using System.Collections.Generic;
4   using System.ComponentModel;
5   using System.Data;
6   using System.Data.SqlTypes;
7   using System.Diagnostics;
8   using System.Drawing;
9   using System.Linq;
10  using System.Security.Claims;
11  using System.Text;
12  using System.Threading.Tasks;
13  using System.Windows.Forms;
14  using System.Windows.Forms.VisualStyles;
15
16  namespace WindowsFormsApp1
17  {
```

System.Collections 추가
System.Collections.Generic 추가

WinForm에 기본적으로 추가되어 있음

컬렉션

(1) 비제네릭 컬렉션 (System.Collections)

- ArrayList, Hashtable, Stack, Queue 등
- 모든 데이터를 object 타입으로 저장 → 성능 저하, 안전성 ↓

(2) 제네릭 컬렉션 (System.Collections.Generic)

- List<T>, Stack<T>, Queue<T>, Dictionary<>
- 형식 안전성 보장 -> 컴파일 시점에서 타입 체크 가능
- 타입을 지정하고 사용할 수 있음 → 성능 ↑, 코드 안전성 ↑

비제네릭 컬렉션 - ArrayList

- 크기를 자동으로 조절할 수 있는 배열 형태 컬렉션.

```
ArrayList list = new ArrayList();

list.Add(100);      // int
list.Add("Hello");  // string
list.Add(3.14);     // double

foreach (object item in list)
{
    Console.WriteLine(item);
}

int num = (int)list[0]; // 명시적 형변환
string str = (string)list[1];
```

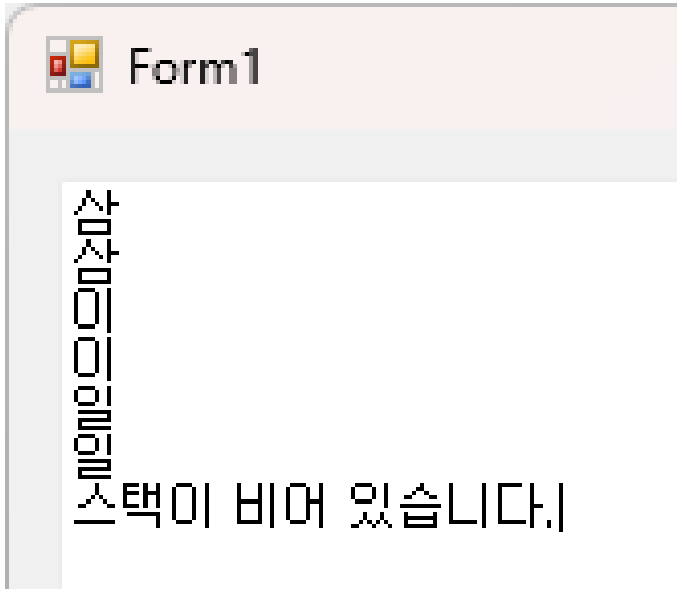
비제네릭 컬렉션 - ArrayList

- ArrayList 주요 메서드

메서드	설명
Add(object value)	요소 추가
Insert(index, value)	특정 위치에 요소 삽입
Remove(value)	해당 요소 제거
RemoveAt(index)	해당 인덱스의 요소 제거
Clear()	전체 요소 삭제
Contains(value)	해당 값이 존재하는지 확인
Sort()	정렬 (단, 같은 타입끼리만 가능)
Reverse()	역순 정렬

비제네릭 컬렉션 - Stack

```
Stack stack = new Stack();

stack.Push("일"); // 데이터 삽입
stack.Push("이");
stack.Push("삼");

try
{
    int stack_size = stack.Count;
    for (int i = 0; i < stack_size + 1; i++)
    {
        // 가장 위 데이터를 삭제하지 않고 사용하기만 함
        textBox_print.Text += $"{stack.Peek()}\r\n";
        // 가장 위 데이터를 사용하고 삭제
        textBox_print.Text += $"{stack.Pop()}\r\n";
    }
}
catch (Exception ex)
{
    textBox_print.Text += ex.Message;
}
```



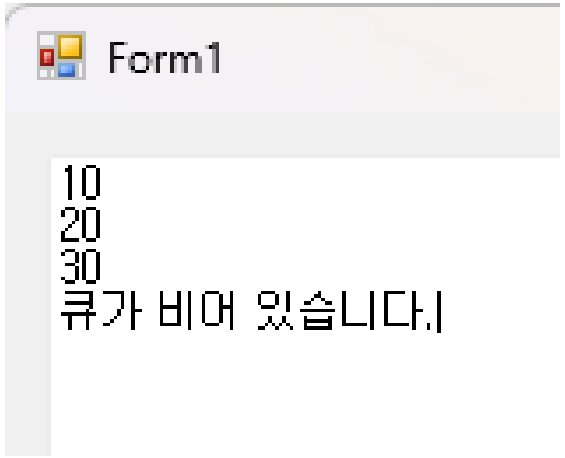
비제네릭 컬렉션 - Stack

- Stack 주요 메서드

메서드	설명
Push(object)	스택의 맨 위에 데이터 추가
Pop()	맨 위 데이터를 꺼내고 제거
Peek()	맨 위 데이터를 꺼내되 제거하지 않음
Clear()	스택 초기화
Count	스택에 들어있는 요소 개수

비제네릭 컬렉션 - Queue

```
Queue queue = new Queue();

queue.Enqueue(10); // 큐에 데이터 삽입
queue.Enqueue(20);
queue.Enqueue(30);

try
{
    int queue_size = queue.Count;
    for(int i = 0; i < queue_size + 1; i++)
    {
        // 먼저 들어간 데이터 부터 사용 후 삭제
        textBox_print.Text += $"{queue.Dequeue()}\r\n";
    }
}
catch (Exception ex)
{
    textBox_print.Text += ex.Message;
}
```



비제네릭 컬렉션 - Queue

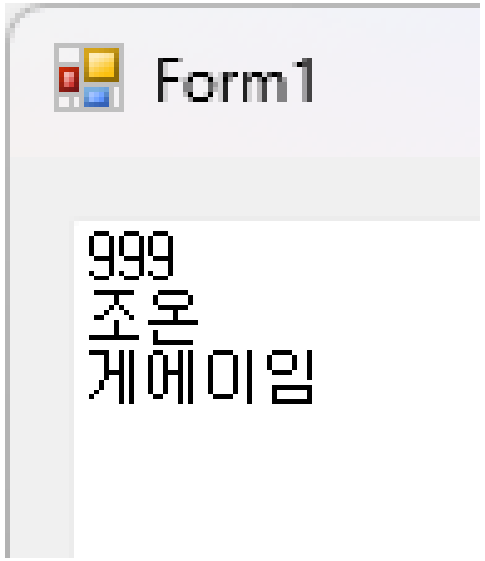
- Queue 주요 메서드

메서드	설명
Enqueue(object)	데이터 추가 (뒤에 넣음)
Dequeue()	데이터 꺼냄 (앞에서 꺼냄 + 제거됨)
Peek()	앞에 있는 데이터 보기 (제거하지 않음)
Clear()	큐 비우기
Count	저장된 항목 수 반환

비제네릭 컬렉션 - Hashtable

- 키(key)와 값(value) 쌍으로 데이터를 저장하는 방식의 컬렉션

```
Hashtable hashtable = new Hashtable();

hashtable[0] = 999;
hashtable["이름"] = "조온";
hashtable["취미"] = "게에이임";

textBox_print.Text += $"{hashtable[0].ToString()}\r\n";
textBox_print.Text += $"{hashtable["이름"]}\r\n";
textBox_print.Text += $"{hashtable["취미"]}\r\n";
```



비제네릭 컬렉션 - Hashtable

- Hashtable 주요 메서드

기능	메서드 또는 속성	설명
추가	Add(key, value)	새 데이터 추가
조회	ht[key] 또는 ht.ContainsKey(key)	키로 값 가져오기
수정	ht[key] = newValue	기존 값 덮어쓰기
삭제	Remove(key)	해당 키와 값 삭제
전체 삭제	Clear()	모든 항목 삭제
전체 개수	Count	저장된 항목 수 반환

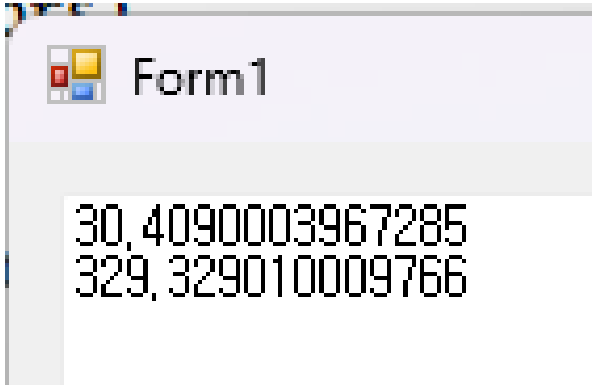
제네릭 컬렉션

- 제네릭(Generic) 이란?
- **데이터 형식을 일반화**하는 문법
- 보통의 컬렉션과 다르게 제네릭은 타입을 선택하여 해당 타입만 추가 가능
- 기능적인 제한이 있지만 안정성이나 속도면에서 제네릭이 유리함

```
Stack stack = new Stack();  // 컬렉션

Stack<int> i_stack = new Stack<int>(); // 제네릭 컬렉션
```

제네릭 컬렉션 - Stack, List

```
Stack<int> i_stack = new Stack<int>();
i_stack.Push(1);
int num = i_stack.Pop();

List<double> d_list = new List<double>();
d_list.Add(30.409f);
d_list.Add(329.329f);

foreach (double d in d_list)
{
    textBox_print.Text += $"{d}\r\n";
}
```



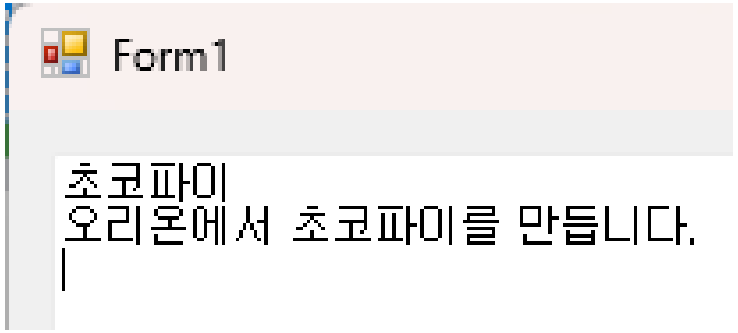
제네릭 컬렉션 - Dictionary

```
// 사용법.1
IDictionary<string, string> data1 = new Dictionary<string, string>();
// 사용법.2
var data2 = new Dictionary<string, string>();

// 데이터 추가
data1.Add("오리온", "고래밥");
data2.Add("농심", "신라면");

// 데이터 제거 및 변경
data1.Remove("오리온");
data2["농심"] = "너구리";
```

```
try
{
    // 같은 인덱스 추가시 오류
    data2.Add("농심", "짜파게티");
}
catch (Exception ex)
{
    textBox_print.Text = ex.Message;
}
```

제네릭 컬렉션 - Dictionary

```
if(!data1.ContainsKey("오리온"))
{
    data1.Add("오리온", "초코파이");
}

if(data1.TryGetValue("오리온", out string value))
{
    textBox_print.Text += $"{value}\r\n";
}

foreach(KeyValuePair<string, string> item in data1)
{
    textBox_print.Text += $"{item.Key}에서 {item.Value}를 만듭니다.\r\n";
}
```



제네릭 컬렉션 - Dictionary

- Dictionary 주요 메서드

메서드 / 속성	설명
Add(key, value)	키와 값을 쌍으로 추가
Remove(key)	키를 기준으로 데이터 제거
ContainsKey(key)	특정 키가 있는지 확인
ContainsValue(value)	특정 값이 있는지 확인
TryGetValue(key, out value)	키에 해당하는 값 안전하게 가져오기
Clear()	모든 항목 제거
Count	저장된 항목 개수

실습. 제네릭 딕셔너리

1. OpenFileDialog를 활용하여 사용자 정보가 아이디,비밀번호,전화번호 순서로 적혀 있는 **accounts.txt** 파일 열기
   - **OpenFileDialog** 사용법 (FileDialog 공식문서 (속성 Tab 참고)) 및 구글 검색
   - 파일 열기는 **StreamReader**를 이용 (StreamReader 공식문서)
   - 2개의 제네릭 딕셔너리를 만들고, 각각 아이디/비밀번호, 아이디/전화번호 데이터를 저장
   - 전화번호가 없다면 null로 저장
2. 로그인 창을 만들고 로그인 성공/실패 메시지 박스 띄우기!
   - 로그인 성공 시 메시지박스에 아이디, 전화번호도 띄우기
   - 실패 시 메시지박스에 실패 이유 보여주기