|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **윈도우프로그래밍\_과제\_6주** | **이름 :** |  | **학번 :** |  |

* **내용 점검**

|  |
| --- |
| 1. 제네릭 클래스 사용 예 |
| class SimpleGeneric<T> {  private T[] values;  private int index;  public SimpleGeneric(int len) { // Constructor  values = new T[len];  index = 0;  }  public void Add(params T[] args){  foreach (T e in args)  values[index++] = e;  }  public void Print(){  foreach (T e in values)  Console.Write(e + " ");  Console.WriteLine();  }  }  class Program{  static void Main(string[] args){  SimpleGeneric<Int32> gInteger = new SimpleGeneric<Int32>(10);  SimpleGeneric<Double> gDouble = new SimpleGeneric<Double>(10);  gInteger.Add(1, 2);  gInteger.Add(1, 2, 3, 4, 5, 6, 7);  gInteger.Add(0);  gInteger.Print();  gDouble.Add(10.0, 20.0, 30.0);  gDouble.Print();  }  } |
| **[실행결과]** |

|  |
| --- |
| 1. 제네릭 메소드 사용 예 |
| class GenericMethodApp{  static void Swap<T>(ref T x, ref T y) {  T temp;  temp = x; x = y; y = temp;  }  public static void Main() {  int a = 1, b = 2; double c = 1.5, d = 2.5;  Console.WriteLine("Before: a = {0}, b = {1}", a, b);  Swap(ref a, ref b);       // 정수형 변수로 호출, 타입 생략 가능            Console.WriteLine(" After: a = {0}, b = {1}", a, b);  Console.WriteLine("Before: c = {0}, d = {1}", c, d);  Swap<double>(ref c, ref d);    // 실수형 변수로 호출            Console.WriteLine(" After: c = {0}, d = {1}", c, d);  }  } |
| **[실행결과]** |

|  |
| --- |
| 1. 제네릭 인터페이스 사용 예 |
| interface IGenericInterface<T> {  void SetValue(T x);  string GetValueType();  }  class GenericClass<T> : IGenericInterface<T> {  private T value;  public void SetValue(T x){  value = x;  }  public String GetValueType(){  return value.GetType().ToString(); //GetType() :자료형 반환  }  }  public class GenericInterfaceApp{  public static void Main(){  GenericClass<Int32> gInteger = new GenericClass<Int32>();  GenericClass<String> gString = new GenericClass<String>();  gInteger.SetValue(10);  gString.SetValue("Text");  Console.WriteLine(gInteger.GetValueType());  Console.WriteLine(gString.GetValueType());  }  } |
| **[실행결과]** |

* 프로그램 과제

1. 교재 260p 5.8

|  |
| --- |
| **프로그램 소스** |
|  |
| **[실행결과]** |

1. 교재 318p 6.11

|  |
| --- |
| **프로그램 소스** |
|  |
| **[실행결과]** |

1. 다음과 같은 인터페이스를 제네릭을 이용하여 다양한 형에서 동작이 가능하도록 구현하는 클래스를 작성하고 테스트 하시오. 단, 스택의 크기는 100.

interface IOperation {

void Insert(string str); //매개변수로 받은 str을 스택에 삽입(push)

string Delete(); //스택 탑 원소 제거(pop) 후 반환

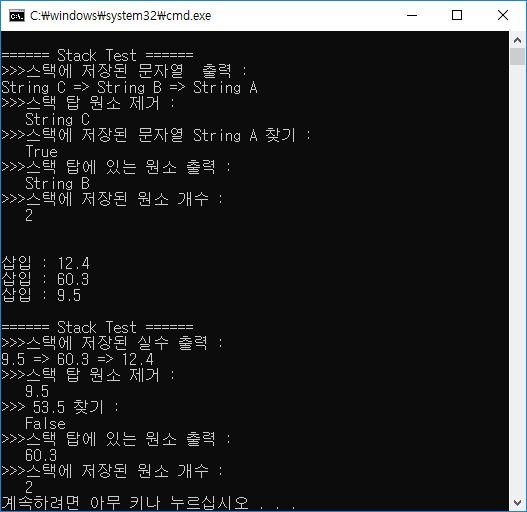
bool Search(string str); //매개변수로 받은 str 원소의 존재 여부 반환

string GetCurrentElt(); //현재 스택 탑에 있는 원소 반환

int NumOfElements(); //스택에 존재하는 원소 개수 반환

void PrintStack(); //스택에 저장된 모든 원소 출력

}



|  |
| --- |
| **프로그램 소스** |
|  |
| **[실행결과]** |

1. 제시된 결과처럼 실행될 수 있도록 3)번 프로그램을 수정하세요.

class Product{

//멤버 변수 : 제목, 연도

//생성자 : 매개변수로 받은 값을 멤버 변수 초기화

//객체 내용을 문자열로 반환하는 ToString() 메소드 재정의

}

class Book //Product 상속

{

//멤버 변수 : 저자,출판사

//생성자 : 매개변수로 받은 값을 멤버 변수초기화

//객체 내용을 문자열로 반환하는 ToString() 메소드 재정의

}

class CD //Product 상속

{

//멤버 변수 : 수록곡

//생성자 : 매개변수로 받은 값을 멤버 변수 초기화

//객체 내용을 문자열로 반환하는 ToString() 메소드 재정의

}

class ProductManage //제네릭 클래스, Product 파생 클래스로 type parameter 제한

{

public ProductManage(int size) // 생성자, 매개변수 값 만큼 type parameter 배열 생성

{

}

public void Insert() // 매개변수로 받은 type parameter의 객체를 배열에 저장

{

}

public void display() // 배열 원소 출력, 매개변수 없음

{

}

}

class Program {

static void Main(string[] args) {

ProductManage<Book> book = new ProductManage<Book>(20);

ProductManage<CD> cd = new ProductManage<CD>(20);

bool flag = true;

while (flag) {

Console.Write("1. Book 2.CD 3.Exit \_\_\_\_\_\_ ");

switch ()

{

case 1:

break;

case 2:

break;

case 3:

flag = false;

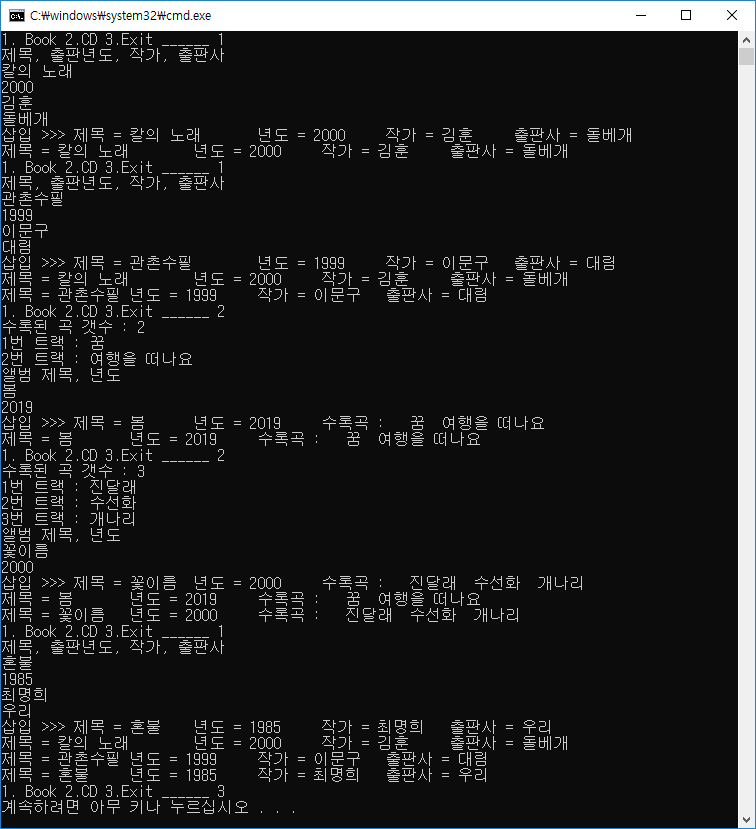
break;

}

}

}

}



|  |
| --- |
| **프로그램 소스** |
|  |
| **[실행결과]** |