|  |  |
| --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования «Полоцкий государственный университет» | |
|  | Факультет информационных технологий  Кафедра технологий программирования |
| Лабораторная работа №3 по курсу «Объектно-ориентированные технологии программирования и стандарты проектирования»  «Наследование»  Вариант 6 | |
| Выполнил | Студент гр. 21-ИТ-1  Катушёнок И.В. |
| Проверил | Ассистент  Хирьянов И.Д. |
| Полоцк, 2023г. | |

**Цель работы:** Цель научиться создавать шаблоны классов, применять шаблоны как для встроенных типов данных, так и для пользовательских классов, понимать особенности применения пользовательских классов в шаблонах.

**Ход работы**

**Задание:**

1. Дополнить и при необходимости модифицировать приложение,

разработанное согласно варианта лабораторной работы No5.

2. Создать шаблон класса, в нем определить конструктор, в котором

реализовано создание массива на заданное количество элементов n

(количество передается в параметре конструктору).

3. В шаблоне определить метод добавления элемента массива и метод

получения элемента массива по индексу.

4. В шаблоне определить метод int findItem(T obj), который ищет элемент в

массиве и возвращает индекс элемента в массиве, а в случае неудачи

возвращает -1. Для работы этого метода потребуется перегрузка

оператора == в классах-наследниках.

5. В шаблоне определить метод T min() и метод T max(), которые

возвращают минимальное и максимальное значение из объектов

массива. Для работы этих методов потребуется перегрузка операторов

сравнения в классах-наследниках. Сравнение объектов производить по

любому из атрибутов класса.

6. В шаблоне определить метод void sort(), который должен отсортировать

массив по возрастанию.

7. В основном теле программы применить шаблон для разных типов

данных: int, char, каждый из классов-наследников.

8. Реализовать все необходимые проверки

9. Дополнить меню программы чтобы можно было продемонстрировать

работу всех функций.

10.Сделать выводы.

**Вариант 6**

Варианты распределяются аналогично предыдущим лабораторным работам.

Листинг 1.1 – Вспомогательный класс для проверки корректности ввода

class CorrectInput

{

public static bool IsOnlyLetters(string str)

{

if (str == "") return false;

if (str == " ") return false;

foreach (char c in str)

{

if (Char.IsNumber(c)) return false;

if (c == '-') return false;

if (c == '.') return false;

if (c == ',') return false;

if (c == ';') return false;

if (c == '/') return false;

if (c == ':') return false;

if (c == '@') return false;

if (c == '#') return false;

if (c == '$') return false;

if (c == '%') return false;

if (c == '^') return false;

if (c == '&') return false;

if (c == '\*') return false;

}

return true;

}

public static bool InRange(int leftRange, int rightRange, int number)

{

if (number >= leftRange && number <= rightRange) return true;

return false;

}

}

Листинг 1.2 – реализация метода Main

class Program

{

static void Main()

{

Catalog catalog = new Catalog();

bool MenuCycle = true;

while (MenuCycle)

{

Console.WriteLine("\tSelect operation\n" +

"[1] - add\n" +

"[2] - info all\n" +

"[3] - delete\n" +

"[4] - edit\n" +

"[5] - search film\n" +

"[6] - sort");

int.TryParse(Console.ReadLine(), out int choise);

switch (choise)

{

case 1:

Console.WriteLine("\tSelect film to add\n" +

"[1] - common\n[2] - favorite\n[3] - blocked");

if(int.TryParse(Console.ReadLine(), out int filmChoise))

switch(filmChoise)

{

case 1:

catalog.AddFilm();

break;

case 2:

catalog.AddFavoriteFilm();

break;

case 3:

catalog.AddBlockedFilm();

break;

}

break;

case 2:

if (catalog.isEmpty())

{

Console.WriteLine("Catalog empty");

break;

}

catalog.SeeAll();

break;

case 3:

catalog.RemoveFromCatalog();

break;

case 4:

catalog.EditFilm();

break;

case 5:

catalog.SearchFilm();

break;

case 6:

catalog.SortFilms();

break;

case -1:

MenuCycle = false;

break;

}

Console.WriteLine("========================");

}

Console.ReadKey();

}

}

Листинг 1.3 – реализация основного класса:

class Catalog

{

List<Film> filmList = new List<Film>();

public void CopyFilm(Film film, int CountOfCopies)

{

for(int i = 0; i < CountOfCopies; i++)

{

Film film1 = new Film(film);

filmList.Add(film1);

}

}

public Film GetFilmFromCatalog(int filmPosition) => filmList[filmPosition];

public void SetFilmInCatalog(int filmPosition, Film film) { filmList[filmPosition] = film; }

public void AddFilm()

{

Console.WriteLine("\tWhich constructor use?\n" +

"[1 - deafult\n[2] - with parametrs\n[3] - copy");

if (int.TryParse(Console.ReadLine(), out int filmChoise))

switch (filmChoise)

{

case 1:

filmList.Add(new Film());

break;

case 2:

filmList.Add(new Film(Film.EnterName(), Film.EnterYear(), Film.EnterType(), Film.EnterRate(), Film.EnterCountry()));

break;

case 3:

if (filmList.Count == 0)

{

Console.WriteLine("catalog empty");

break;

}

Console.WriteLine("Enter film number");

int.TryParse(Console.ReadLine(), out int filmPosition);

if(CorrectInput.InRange(0, filmList.Count - 1, filmPosition)) filmList.Add(new Film(filmList[filmPosition]));

else

{

Console.WriteLine("Incorrect film number");

break;

}

break;

}

}

public void AddBlockedFilm()

{

Console.WriteLine("\tWhich constructor use?\n" +

"[1 - deafult\n[2] - with parametrs\n[3] - copy");

if (int.TryParse(Console.ReadLine(), out int filmChoise))

switch (filmChoise)

{

case 1:

filmList.Add(new BlockedFilm());

break;

case 2:

filmList.Add(new BlockedFilm(Film.EnterName(), Film.EnterYear(), Film.EnterType(), Film.EnterRate(), Film.EnterCountry(), BlockedFilm.EnterDescription()));

break;

case 3:

if (filmList.Count == 0)

{

Console.WriteLine("catalog empty");

break;

}

Console.WriteLine("Enter film number");

int.TryParse(Console.ReadLine(), out int filmPosition);

if (CorrectInput.InRange(0, filmList.Count - 1, filmPosition))

{

if (filmList[filmPosition] is BlockedFilm)

{

filmList.Add(new BlockedFilm((BlockedFilm)filmList[filmPosition]));

}

else filmList.Add(new Film(filmList[filmPosition]));

}

else

{

Console.WriteLine("Incorrect film number");

break;

}

break;

}

}

public void AddFavoriteFilm()

{

Console.WriteLine("\tWhich constructor use?\n" +

"[1 - deafult\n[2] - with parametrs\n[3] - copy");

if (int.TryParse(Console.ReadLine(), out int filmChoise))

switch (filmChoise)

{

case 1:

filmList.Add(new FavoriteFilm());

break;

case 2:

filmList.Add(new FavoriteFilm(Film.EnterName(), Film.EnterYear(), Film.EnterType(), Film.EnterRate(), Film.EnterCountry(), FavoriteFilm.EnterDescription()));

break;

case 3:

if (filmList.Count == 0)

{

Console.WriteLine("catalog empty");

break;

}

Console.WriteLine("Enter film number");

int.TryParse(Console.ReadLine(), out int filmPosition);

if (CorrectInput.InRange(0, filmList.Count - 1, filmPosition))

{

if (filmList[filmPosition] is FavoriteFilm)

{

filmList.Add(new FavoriteFilm((FavoriteFilm)filmList[filmPosition]));

}

else filmList.Add(new Film(filmList[filmPosition]));

}

else

{

Console.WriteLine("Incorrect film number");

break;

}

break;

}

}

public void EditFilm()

{

if (filmList.Count == 0)

{

Console.WriteLine("catalog empty");

return;

}

Console.WriteLine("Enter film number");

int.TryParse(Console.ReadLine(), out int filmPosition);

if (CorrectInput.InRange(0, filmList.Count - 1, filmPosition)) filmList[filmPosition].Edit();

else Console.WriteLine("Incorrect input");

}

public void RemoveFromCatalog()

{

Console.Write("Enter film number: ");

int.TryParse(Console.ReadLine(), out int position);

if (filmList.Count == 0)

{

Console.WriteLine("Catalog is empty");

return;

}

if (CorrectInput.InRange(0, filmList.Count - 1, position))

{

filmList.RemoveAt(position);

GC.Collect();

}

else

{

Console.WriteLine("Film with this number not found");

return;

}

}

public void GetInfoFromPosition(int filmPosition) { filmList[filmPosition].SeeInfo(); }

public void SearchFilm()

{

if (isEmpty())

{

Console.Write("Catalog empty");

return;

}

Console.WriteLine("How you want to search?");

Console.WriteLine("[1] - Name\n[2] - Year\n[3] - Type\n[4] - Rating\n[5] - Country.");

int.TryParse(Console.ReadLine(), out int choise);

switch (choise)

{

case 1:

SearchName();

break;

case 2:

SearchYear();

break;

case 3:

SearchType();

break;

case 4:

SeatchRate();

break;

case 5:

SearchCounrty();

break;

}

}

public void SearchCounrty()

{

Console.WriteLine("Enter country");

string Country = Console.ReadLine();

for (int i = 0; i < filmList.Count; i++)

{

if (filmList[i].GetCountry() == Country)

{

Console.Write(i + ".");

filmList[i].SeeInfo();

}

}

}

public void SearchType()

{

Console.WriteLine("Enter type");

string Type = Console.ReadLine();

for (int i = 0; i < filmList.Count; i++)

{

if (filmList[i].GetType() == Type)

{

Console.Write(i + ".");

filmList[i].SeeInfo();

}

}

}

public void SearchName()

{

Console.WriteLine("Enter name");

string Name = Console.ReadLine();

for (int i = 0; i < filmList.Count; i++)

{

if (filmList[i].GetName().IndexOf(Name) != -1)

{

Console.Write(i + ".");

filmList[i].SeeInfo();

}

}

}

public void SearchYear()

{

Console.WriteLine("Enter year");

int.TryParse(Console.ReadLine(), out int year);

Console.WriteLine("[1] - >\n[2] - <");

int.TryParse(Console.ReadLine(), out int choise);

for (int i = 0; i < filmList.Count; i++)

{

if (filmList[i].GetYear() > year && choise == 1)

{

Console.Write(i + ".");

filmList[i].SeeInfo();

}

if (filmList[i].GetYear() < year && choise == 2)

{

Console.Write(i + ".");

filmList[i].SeeInfo();

}

}

}

public void SeatchRate()

{

Console.WriteLine("Enter search rating(0-100)");

int rate;

while(true)

{

int.TryParse(Console.ReadLine(), out rate);

if (rate < 100 && rate > 0) break;

Console.WriteLine("Incorrect input, try again");

}

Console.WriteLine("[1] - <\n[2] - >");

int chose = Convert.ToInt32(Console.ReadLine());

foreach (Film film in filmList)

{

if(film.GetRate() > rate && chose == 1) film.SeeInfo();

if(film.GetRate() > rate && chose == 2) film.SeeInfo();

}

}

public void SeeAll()

{

Console.WriteLine("========================");

for (int i = 0; i < filmList.Count; i++)

{

Console.Write(i + ".");

//filmList[i].SeeInfo();

if (filmList[i] is FavoriteFilm)

{

((FavoriteFilm)filmList[i]).SeeInfo();

}

else if (filmList[i] is BlockedFilm)

{

((BlockedFilm)filmList[i]).SeeInfo();

}

else

{

filmList[i].SeeInfo();

}

Console.WriteLine(" ");

}

}

public bool isEmpty()

{

if (filmList.Count == 0) return true;

else return false;

}

public void SortFilms()

{

int choise = 0;

Console.WriteLine("How you want to sort?");

Console.WriteLine("[1] - Year\n[2] - Rating.");

try

{

choise = Convert.ToInt32(Console.ReadLine());

}

catch

{

Console.WriteLine("Incorrect input");

return;

}

switch (choise)

{

case 1:

for (int i = 0; i < filmList.Count; i++)

for (int j = 0; j < filmList.Count; j++)

{

if (filmList[i].GetYear() < filmList[j].GetYear())

{

Film temp = filmList[i];

filmList[i] = filmList[j];

filmList[j] = temp;

}

}

break;

case 2:

for (int i = 0; i < filmList.Count; i++)

for (int j = 0; j < filmList.Count; j++)

{

if (filmList[i].GetRate() < filmList[j].GetRate())

{

Film temp = filmList[i];

filmList[i] = filmList[j];

filmList[j] = temp;

}

}

break;

}

}

}

Листинг 1.4 – реализация дополнительного класса:

class Film

{

protected string \_name;

protected int \_year;

protected int \_rate;

protected string \_type;

protected string \_country;

public Film()

{

Console.WriteLine("Constructor without parametrs is called");

SetName(EnterName());

SetYear(EnterYear());

SetType(EnterType());

SetRate(EnterRate());

SetCountry(EnterCountry());

}

public Film(Film film)

{

\_name = film.\_name;

\_year = film.\_year;

\_type = film.\_type;

\_rate = film.\_rate;

\_country = film.\_country;

Console.WriteLine("Copy constructor was called");

}

public Film(string name, int year, string type, int rate, string country)

{

Console.WriteLine("Constructor with parametrs was called");

\_name = name;

\_year = year;

\_type = type;

\_rate = rate;

\_country = country;

}

~Film()

{

Console.WriteLine("Destructor was called");

}

public string GetName() => \_name;

public int GetYear() => \_year;

public int GetRate() => \_rate;

public string GetType() => \_type;

public string GetCountry() => \_country;

public void SetName(string name) { \_name = name; }

public void SetYear(int year) { \_year = year; }

public void SetType(string type) {\_type = type; }

public void SetRate(int rate) { \_rate = rate; }

public void SetCountry(string country) { \_country = country; }

public static string EnterName()

{

Console.Write("Enter film name: ");

string name = Console.ReadLine();

while(true)

{

if(name != "")

{

return name;

}

}

}

public static int EnterYear()

{

while (true)

{

Console.Write("Enter film year: ");

int.TryParse(Console.ReadLine(), out int year);

if (CorrectInput.InRange(1927, 2023, year))

{

return year;

}

Console.WriteLine("Incorrect input, try again");

}

}

public static string EnterType()

{

while (true)

{

Console.Write("Enter film type: ");

string type = Console.ReadLine();

if (CorrectInput.IsOnlyLetters(type))

{

return type;

}

Console.WriteLine("Incorrect input, try again");

}

}

public static int EnterRate()

{

while (true)

{

Console.Write("Enter film rating(0-100): ");

bool check = int.TryParse(Console.ReadLine(), out int rate);

if (CorrectInput.InRange(0, 100, rate) && check)

{

return rate;

}

Console.WriteLine("Incorrect input, try again");

}

}

public static string EnterCountry()

{

while (true)

{

Console.Write("Enter film country: ");

string country = Console.ReadLine();

if (CorrectInput.IsOnlyLetters(country))

{

return country;

}

Console.WriteLine("Incorrect input, try again");

}

}

public void Edit()

{

Console.WriteLine("What do you want to change?");

Console.WriteLine("[1] - Name\n[2] - Year\n[3] - Type\n[4] - Rating\n[5] - Country.");

int edit = Convert.ToInt32(Console.ReadLine());

switch (edit)

{

case 1:

SetName(EnterName());

break;

case 2:

SetYear(EnterYear());

break;

case 3:

SetType(EnterType());

break;

case 4:

SetRate(EnterRate());

break;

case 5:

SetCountry(EnterCountry());

break;

}

}

public void SeeInfo()

{

Console.WriteLine("\tFilm: " + \_name);

Console.WriteLine("\tYear: " + \_year);

Console.WriteLine("\tType: " + \_type);

Console.WriteLine("\tRating: " + \_rate);

Console.WriteLine("\tCountry: " + \_country);

}

Листинг 1.5 – реализация класса наслендика FavoriteFilm

class FavoriteFilm : Film

{

private string \_description;

public FavoriteFilm()

{

Console.WriteLine("class FavoriteFilm constructor was called");

SetDescription(EnterDescription());

}

public FavoriteFilm(string name, int year, string type, int rate, string country, string description) : base(name, year , type, rate, country)

{

\_description = description;

Console.WriteLine("class FavoriteFilm constructor with paramerts was called");

}

public FavoriteFilm(FavoriteFilm favoriteFilm) : base(favoriteFilm)

{

\_description = favoriteFilm.GetDescription();

Console.WriteLine("class FavoriteFilm copy constructor was called");

}

~FavoriteFilm()

{

Console.WriteLine("class FavoriteFilm destructor was called");

}

public static string EnterDescription()

{

while (true)

{

Console.Write("Enter favorite film description: ");

string description = Console.ReadLine();

if (CorrectInput.IsOnlyLetters(description))

{

return description;

}

Console.WriteLine("Incorrect input, try again");

}

}

public void SetDescription(string description)

{

\_description = description;

}

public string GetDescription()

=> \_description;

public new void SeeInfo()

{

Console.WriteLine("\tFilm: " + GetName());

Console.WriteLine("\tYear: " + GetYear());

Console.WriteLine("\tType: " + GetType());

Console.WriteLine("\tRating: " + GetRate());

Console.WriteLine("\tCountry: " + GetCountry());

Console.WriteLine("\tDescription: " + \_description);

}

}

Листинг 1.6 – реализация класса наследника BlockedFilm

class BlockedFilm : Film

{

private string \_blockReason;

public BlockedFilm()

{

Console.WriteLine("class BlockedFilm constructor was called");

SetDescription(EnterDescription());

}

public BlockedFilm(string name, int year, string type, int rate, string country, string blockReason) : base(name, year, type, rate, country)

{

SetDescription(blockReason);

Console.WriteLine("class BlockedFilm constructor with parametrs was called");

}

public BlockedFilm(BlockedFilm blockedFilm) : base(blockedFilm)

{

\_blockReason = blockedFilm.\_blockReason;

Console.WriteLine("class BlockedFilm copy constructor was called");

}

~BlockedFilm()

{

Console.WriteLine("class BlockedFilm destructor was called");

}

public static string EnterDescription()

{

while (true)

{

Console.Write("Enter block reason: ");

string blockReason = Console.ReadLine();

if (CorrectInput.IsOnlyLetters(blockReason))

{

return blockReason;

}

Console.WriteLine("Incorrect input, try again");

}

}

public void SetDescription(string blockReason) { \_blockReason = blockReason; }

public string GetDescription() => \_blockReason;

public new void SeeInfo()

{

Console.WriteLine("\tFilm: " + GetName());

Console.WriteLine("\tYear: " + GetYear());

Console.WriteLine("\tType: " + GetType());

Console.WriteLine("\tRating: " + GetRate());

Console.WriteLine("\tCountry: " + GetCountry());

Console.WriteLine("\tBlock reason: " + \_blockReason);

}

}

class Template <T>

{

private T[] TArray;

private int ArrayLength;

public Template(int n)

{

TArray = new T[n];

ArrayLength = n;

}

public int GetArrayLength()

{

return ArrayLength;

}

public T GetElement(int n)

{

return TArray[n];

}

public T[] GetArray()

{

return TArray;

}

public T GetFromIndex(int index)

{

return TArray[index];

}

public void SetByIndex(int index, T element)

{

if (TArray.Length == 0)

{

Console.WriteLine("Array is empty");

return;

}

if (CorrectInput.InRange(0, TArray.Length - 1, index))

TArray[index] = element;

else

Console.WriteLine("Index was outside of array");

}

public int findItem(Template<T> template ,T obj)

{

return template == obj;

}

public void Sort()

{

List<T> list = new List<T>(TArray);

list.Sort();

TArray = list.ToArray();

}

public T max()

{

List<T> list = new List<T>(TArray);

list.Sort();

return list[list.Count - 1];

}

public T min()

{

List<T> list = new List<T>(TArray);

list.Sort();

return list[0];

}

static public int operator == (Template<T> template, T secondElement)

{

for (int i = 0; i < template.ArrayLength; i++)

{

if (Object.Equals(template.TArray[i], secondElement))

return i;

}

return -1;

}

static public int operator !=(Template<T> template, T secondElement)

{

return -1;

}

static public void SeeArrays(Template<int> intArray, Template<char> charArray,

Template<FavoriteFilm> favoriteFilmArray, Template<BlockedFilm> blockedFilmArray)

{

Console.Write("Int Array: ");

for (int i = 0; i < intArray.GetArrayLength(); i++)

{

Console.Write(intArray.GetFromIndex(i) + ", ");

}

Console.WriteLine("");

Console.Write("Char Array: ");

for (int i = 0; i < charArray.GetArrayLength(); i++)

{

Console.Write(charArray.GetFromIndex(i) + ", ");

}

Console.WriteLine("");

Console.WriteLine("\t\tfavorite films Array:");

for (int i = 0; i < favoriteFilmArray.GetArrayLength(); i++)

{

favoriteFilmArray.GetFromIndex(i).SeeInfo();

Console.WriteLine("");

}

Console.WriteLine("\t\tblocked films Array:");

for (int i = 0; i < blockedFilmArray.GetArrayLength(); i++)

{

blockedFilmArray.GetFromIndex(i).SeeInfo();

Console.WriteLine("");

}

}

static public bool templateMenu(Template<int> intArray, Template<char> charArray,

Template<FavoriteFilm> favoriteFilmArray, Template<BlockedFilm> blockedFilmArray)

{

Console.WriteLine("\tSelect template operation\n" +

"[1] - set array element\n" +

"[2] - max array element\n" +

"[3] - min array element\n" +

"[4] - get array element\n" +

"[5] - sort array\n" +

"[6] - other operations\n" +

"[7] - see all arrays\n" +

"[8] - find element");

//"[6] - see arrays");

if (int.TryParse(Console.ReadLine(), out int templateChoise))

{

switch (templateChoise)

{

case 1:

Console.WriteLine("\tSelect array\n" +

"[1] - Integer array\n" +

"[2] - Char array\n" +

"[3] - Favorite film array\n" +

"[4] - Blocked film array\n");

if (int.TryParse(Console.ReadLine(), out int setArrayTypeChoise))

{

Console.WriteLine("Enter element index");

int.TryParse(Console.ReadLine(), out int index);

switch (setArrayTypeChoise)

{

case 1:

Console.WriteLine("Enter number");

int.TryParse(Console.ReadLine(), out int intElement);

intArray.SetByIndex(index, intElement);

return true;

case 2:

Console.WriteLine("Enter symbol");

char.TryParse(Console.ReadLine(), out char charElement);

charArray.SetByIndex(index, charElement);

return true;

case 3:

FavoriteFilm film = new FavoriteFilm();

favoriteFilmArray.SetByIndex(index, film);

return true;

case 4:

BlockedFilm blockedFilm = new BlockedFilm();

blockedFilmArray.SetByIndex(index, blockedFilm);

return true;

}

}

return true;

case 2:

Console.WriteLine($"Maximum element in integer array: {intArray.max()}");

Console.WriteLine($"Maximum elemnt in char array: {charArray.max()}");

Console.WriteLine($"Maximum element in favoriteFilmArray array:");

favoriteFilmArray.max().SeeInfo();

Console.WriteLine($"Maximum element in blockedFilmArray array:");

blockedFilmArray.max().SeeInfo();

return true;

case 3:

Console.WriteLine($"Minimum element in integer array: {intArray.min()}");

Console.WriteLine($"Minimum element in char array: {charArray.min()}");

Console.WriteLine("Minimum element in favoriteFilmArray array:");

favoriteFilmArray.min().SeeInfo();

Console.WriteLine("Minimum element in blockedFilmArray array:");

blockedFilmArray.min().SeeInfo();

return true;

case 4:

Console.WriteLine("\tSelect array\n" +

"[1] - Integer array\n" +

"[2] - Char array\n" +

"[3] - Favorite film array\n" +

"[4] - Blocked film array\n");

if (int.TryParse(Console.ReadLine(), out int getArrayTypeChoise))

{

Console.WriteLine("Enter element index");

int.TryParse(Console.ReadLine(), out int index);

switch (getArrayTypeChoise)

{

case 1:

if (CorrectInput.InRange(0, intArray.GetArrayLength() - 1, index))

Console.WriteLine(intArray.GetFromIndex(index));

else

Console.WriteLine("Incorrect array index");

return true;

case 2:

if (CorrectInput.InRange(0, charArray.GetArrayLength() - 1, index))

Console.WriteLine(charArray.GetFromIndex(index));

else

Console.WriteLine("Incorrect array index");

return true;

case 3:

if (CorrectInput.InRange(0, favoriteFilmArray.GetArrayLength() - 1, index))

favoriteFilmArray.GetElement(index).SeeInfo();

else

Console.WriteLine("Incorrect array index");

return true;

case 4:

if (CorrectInput.InRange(0, blockedFilmArray.GetArrayLength() - 1, index))

blockedFilmArray.GetElement(index).SeeInfo();

else

Console.WriteLine("Incorrect array index");

return true;

}

}

return true;

case 5:

intArray.Sort();

charArray.Sort();

favoriteFilmArray.Sort();

blockedFilmArray.Sort();

Console.WriteLine("Sorted arrays: ");

Template<int>.SeeArrays(intArray, charArray, favoriteFilmArray, blockedFilmArray);

return true;

case 6:

return false;

case 7:

Template<int>.SeeArrays(intArray, charArray, favoriteFilmArray, blockedFilmArray);

return true;

case 8:

Console.WriteLine("\tSelect array\n" +

"[1] - Integer array\n" +

"[2] - Char array\n" +

"[3] - Favorite film array\n" +

"[4] - Blocked film array\n");

if (int.TryParse(Console.ReadLine(), out int findChoise))

switch (findChoise)

{

case 1:

Console.WriteLine("Enter number");

int.TryParse(Console.ReadLine(), out int number);

Console.WriteLine(intArray.findItem(intArray, number));

return true;

case 2:

Console.WriteLine("Enter symbol");

char.TryParse(Console.ReadLine(), out char symbol);

Console.WriteLine(charArray.findItem(charArray, symbol));

return true;

case 3:

Console.WriteLine( favoriteFilmArray.findItem(favoriteFilmArray, new FavoriteFilm()) );

return true;

case 4:

Console.WriteLine( blockedFilmArray.findItem(blockedFilmArray, new BlockedFilm()) );

return true;

}

return true;

}

}

return true;

}

}

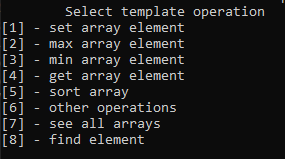
} 

Рисунок 1. Начальное меню

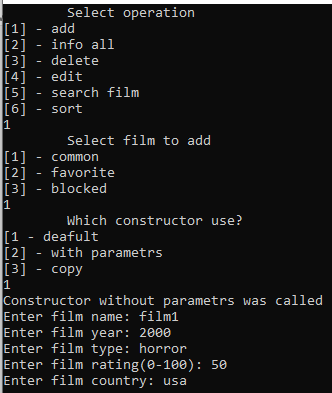


Рисунок 2. Добавление элемента родительского класса

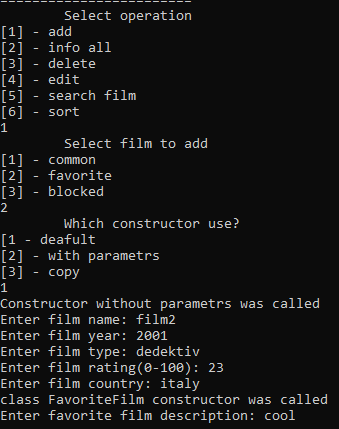


Рисунок 3. Добавление элемента первого дочернего класса с помощью конструктора без параметров

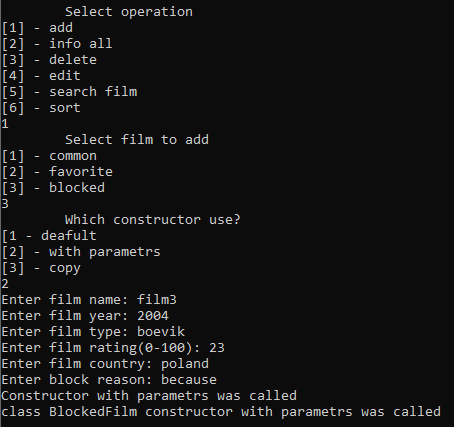


Рисунок 4. Добавление элемента второго дочернего класса с помощью конструктора с параметрами



Рисунок 5. Просмотр всех фильмов

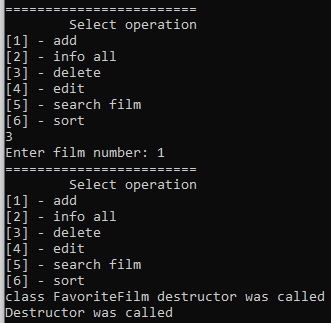


Рисунок 5. Удаление фильма

**Вывод:** В ходе данный лабораторной работе было изучено как создавать иерархию, создавать объекты базовых классов и классов-наследников. вызывать методы базового класса и классов наследников. Влияние модификаторов доступа при наследовании, порядок вызова конструкторов и деструкторов при наследовании.