Лабораторна робота №2

Побудова за допомогою програми Rational Rose UML діаграми класів автоматизованої інформаційної системи.

***Мета роботи:*** *засвоїти методику та виробити практичні в побудові за допомогою програми Rational Rose UML діаграми класів автоматизованої інформаційної системи.*

Теоретичні відомості

Діаграма класів

Діаграма класів – це один з найбільш часто використовуваних видів діаграм UML. Звичайне створення діаграми класів знаменує собою закінчення процесу аналізу і початок процесу проектування.

Клас на діаграмі зображується у вигляді прямокутника, розділеного горизонтальними лініями на три частини. У першій частині вказується назва класу. Як правило, ім'я класу складається з одного, максимум двох слів. Друга частина містить перелік властивостей класу, які характеризують той чи інший об'єкт цього класу в моделі предметної області. Третя частина містить перелік методів (операцій), що відображають його поведінку в моделі предметної області (рис.1.).

У UML властивості та методи з модифікаторами (рівнями) доступу позначаються спеціальними символами ліворуч від їхніх імен:

|  |  |
| --- | --- |
| Символ | Значення |
| + | Public |
| - | Private |
| # | Protected |



Рис 1 UML діаграма класів на якому зображено члени класу Emploe.

Крім внутрішньої структури класів на діаграмі класів вказуються різні відносини між класами. Базовими відносинами або зв'язками є:

1. відношення асоціації;
2. відношення узагальнення;
3. відношення залежності;
4. відношення агрегації.

Відношення асоціації означає наявність зв'язку між класами або об’єктами, наприклад, клас Студент асоційований з класом Інститут. Асоціація може мати ім'я, наприклад, Навчається. Поруч із іменем асоціації звичайно ставлять стрілку, що вказує напрям читання імені («Студент навчається в інституті»).

Зв'язок між екземплярами класів має на увазі деякі ролі, які відповідні об'єкти грають по відношенню один до одного. Роль пов'язана з напрямком асоціації. Так, стосовно студентів інститут – організація, що здійснює іхнє навчання, тобто роль інституту можна назвати Місце навчання. Студент для інституту – об'єкт навчальної діяльності інституту, тобто Той, якого навчають. Якщо роль власного імені немає, то можна вважати, що її ім'я збігається з ім'ям класу, стосовно якого визначається ця роль. Для розглянутого прикладу це відповідно ролі Студент та Інститут (рис.2. а), але роль можна вказати і явно (рис 2. б).



Рис 2. UML-діаграма класів, на якій відображається відношення асоціації: а – із вказівкою іменем асоціації і її напрямків; б – із вказівкою імен ролей; в – із вказівкою множинності.

Роль також має характеристику множинності, що показує скільки об'єктів може брати участь в одному зв'язку з кожної сторони. Допускається вказувати множинність:

\*-від 0 до безкінченності;

<ціле> ..\* - від заданого числа до безкінченності;

<ціле> - точна певна кількість об'єктів;

<ціле1>,<ціле2> - кілька варіантів точної кількості об'єктів;

<ціле1>..<ціле2> - діапазон об'єктів.

Відношення узагальнення (або, як часто говорять, наслідування) на діаграмах позначається дуже просто - незамальованою трикутною стрілкою, направленою на батьківський клас.



Рис.3. UML-діаграма класів, на якій відображається відношення узагальнення.

Відношення залежності виникає тоді, коли реалізація класу одного об'єкту залежить від специфікації операцій класу іншого об'єкту. І якщо зміниться специфікація операцій цього класу, нам неминуче доведеться вносити зміни і в залежний клас. Розглянемо приклад, деколи нам в руки потрапляють фільми, які одразу невдається запустити з медіаплеєру, тому що на комп'ютері не встановлений відповідний кодек. Якщо специфікація операцій зміниться, то доведеться міняти код медіаплеєру, інакше він просто незможе працювати з якимось кодеком, і в кращому випадку завершить свою роботу з помилкою. На рисунку 4. зображено відношення залежності.



Рис. 4. UML-діаграма класів, на якій відображається відношення залежності.

У реальності асоціація буває вкрай рідко. Зазвичай при найближчому розгляді під асоціацією розуміється складніше відношення між класами, наприклад, зв'язок типу “ціле-частини”. Такий вигляд асоціації називається асоціацією з агрегацією. В цьому випадку один клас має вищий статус (ціле) і складається з нижчих по статусу класів (частин). При цьому виділяють просту і композитну агрегацію і говорять про власне агрегації і композиції. Проста агрегація передбачає, що частини, відокремлені від цілого, можуть продовжувати своє існування незалежно від нього. Під композитною ж агрегацією розуміється ситуація, коли ціле володіє своїми частинами і їх час життя відповідає часу життя цілого, тобто незалежно від цілого частини класів існувати не можуть. На рисунку 5. зображено відношення простої та композитної агрегацій.



Рис. 5. UML-діаграма класів, на якій відображається відношення асоціації: а – проста агрегація; б-композитна агрегація.

Активізувати робоче вікно діаграми класів можна кількома способами:

* вікно діаграми класів з'являється за замовчуванням в робочому вікні діаграми після створення нового проекту;
* клацнути на кнопці з зображенням діаграми класів на стандартній панелі інструментів;
* розкрити логічне подання (Logical View) в браузері проекту і двічі клацнути на піктограмі Main (Головна);
* виконати операцію головного меню: Browse/Class Diagram (Обзор/Діаграмма класів).

При цьому з'являється нове вікно з чистим робочим листом діаграми класів і спеціальна панель інструментів, що містить кнопки із зображенням графічних примітивів, необхідних для розробки діаграми класів (Табл. 1).

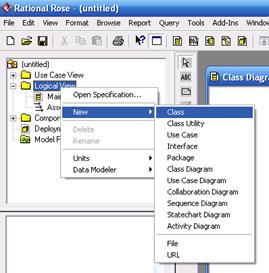
Призначення окремих кнопок панелі можна дізнатися також з спливаючих підказок.

Таблиця 1 Призначення кнопок спеціальної панелі інструментів для діаграми класів

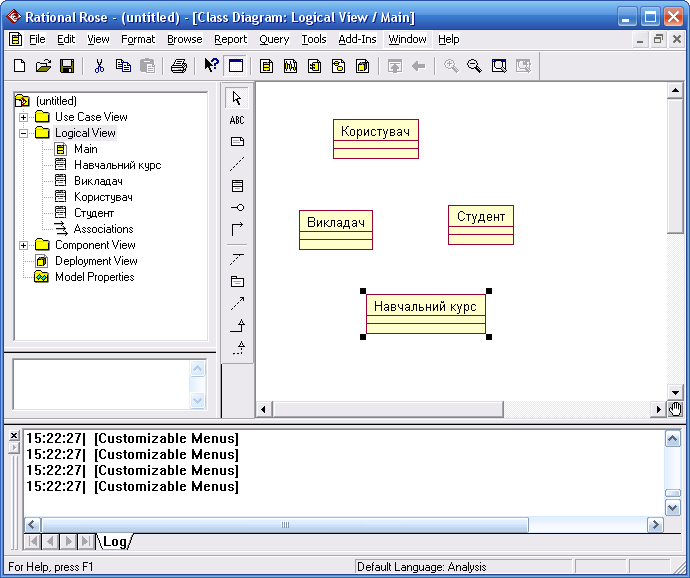
|  |  |  |
| --- | --- | --- |
| Графічне зображення | Випливаюча підказка | Призначення кнопки |
|  | Selection Tool | Перетворює зображення курсору у форму стрілки для подальшого виділення елементів на діаграмі |
|  | Text Box | Додає на діаграму текстову область |
|  | Note | Додає до діаграми примітку |
|  | Anchor Note to Item | Додає на діаграму зв'язок примітки з відповідним графічним елементом діаграми |
|  | Class | Додає на діаграму клас |
|  | Interface | Додає на діаграму інтерфейс |
|  | Unidirectional Association | Додає на діаграму спрямовану асоціацію |
|  | Association Class | Додає на діаграму асоціацію клас |
|  | Package | Додає на діаграму пакет |
|  | Dependency or Instantiates | Додає на діаграму відношення залежності |
|  | Generalization | Додає на діаграму відношення узагальнення |
|  | Realize | Додає на діаграму відношення реалізації |

На спеціальній панелі інструментів за замовчуванням присутня тільки частина піктограм елементів , які можуть бути використані для побудови діаграми класів . Додати кнопки з піктограмами інших графічних елементів або видалити непотрібні кнопки можна за допомогою налаштування спеціальної панелі інструментів. Відповідне діалогове вікно налаштування спеціальної панелі інструментів для діаграми класів можна викликати за допомогою операції контекстного меню Customize (Налаштування).

Для додавання класу на діаграму класів потрібно за допомогою лівої кнопки миші натиснути кнопку із зображенням піктограми класу на спеціальній панелі інструментів, відпустити ліву кнопку миші і клацнути лівою кнопкою миші на вільному місці робочого листа діаграми. На діаграмі з'явиться зображення класу з маркерами зміни його геометричних розмірів і запропонованим середовищем ім'ям за замовчуванням NewClass . Або створити клас через пункт спливаючого меню New- > Class (Рис.1) , а потім " перетягнути " створений клас з вікна браузера проекту в область вікна діаграми класів ( Мал. 1.4).



Розглянемо приклад автоматизованої інформаційної системи реєстрації навчальних курсів в якому потрібно створити 4-ри класи: Користувач, Викладач, Студент та Навчальний курс



З усіх графічних елементів середовища Rational Rose клас володіє максимальним набором властивостей, головними з яких є його атрибути і операції. До речі , саме діаграма класів використовується в середовищі Rational Rose для генерації програмного коду

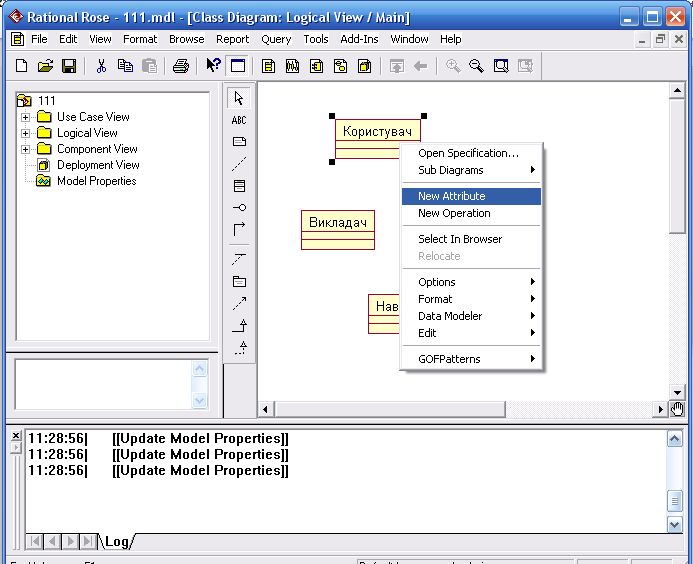
Додамо аттрібути для класу "Користувач":

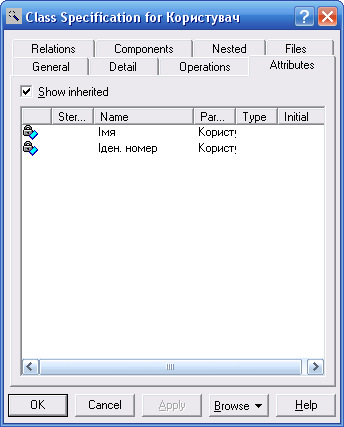
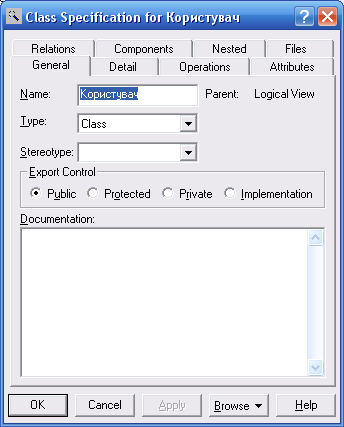
* Ім'я ;
* Ід . номер.

Додати атрибут до створеного раніше класу можна одним з таких способів:

За допомогою операції контекстного меню New Attribute (Новий атрибут) для класу , виділеного на діаграмі класів .

За допомогою операції контекстного меню: NewAttribute (НовийАтрібут) для класу , виділеного в браузері проекту.





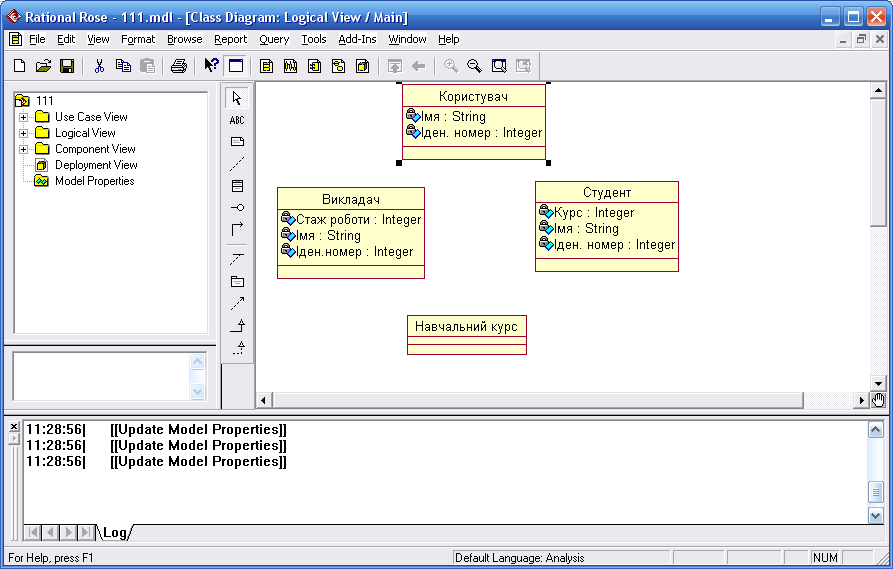


Після додавання атрибута до класу за замовчуванням йому присвоюється ім'я name і деякий квантор видимості.

Видимість атрибутів на діаграмі класів зображується у формі спеціальних піктограм. Використовувані піктограми видимості зображуються перед ім'ям відповідного атрибута і мають наступний сенс.

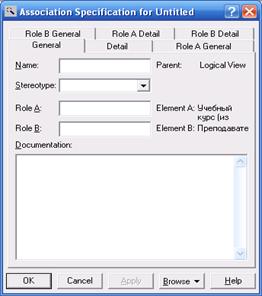
|  |  |  |
| --- | --- | --- |
| Графічне зображення | Назва | Призначення |
|  | Public | Загальнодоступний або відкритий. У нотації мови UML такому атрибуту відповідає знак «+» |
|  | Protected | Захищений. У нотації мови UML такому атрибуту відповідає знак «#» |
|  | Private | Закритий. У нотації мови UML такому атрибуту відповідає знак «-» |
|  | Implementation | Реалізація. У нотації мови UML такому атрибуту відповідає знак «~» |

Тепер додамо атрибути всіх класах, зображеним у вікні діаграми класів

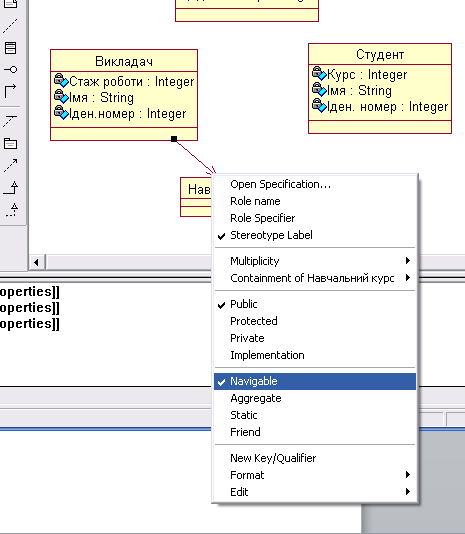


Створюємо відношення асоціації. Змінимо властивості для даної асоціації, запропоновані середовищем за замовчуванням. Це можна виконати за допомогою вікна специфікації властивостей асоціації. Доступ до діалогового вікна специфікації властивостей асоціації Association Specification можна отримати після виділення лінії асоціації на діаграмі класів або в браузері проекту і подвійного клацання на ній лівої кнопки миші.





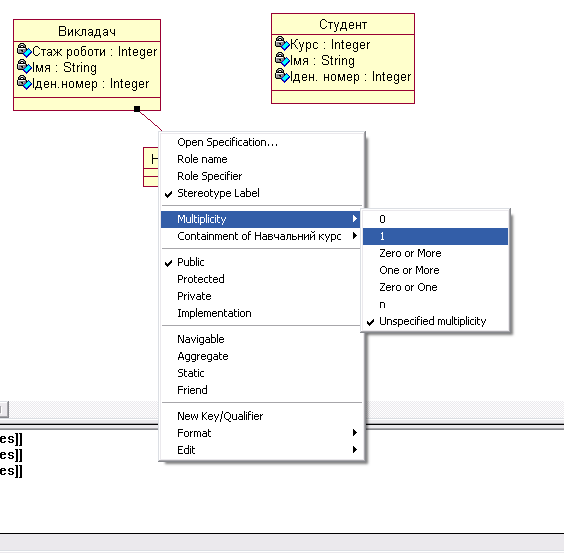
Перше, що ми робимо - прибираємо спрямованість відносини: у спливаючому меню асоціації («Клік» правою кнопкою миші) прибираємо галочку з пункту «Navigate»



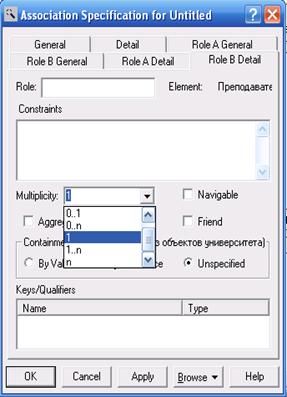
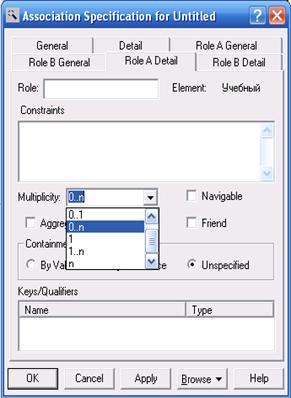
Другим кроком потрібно вказати кратності для асоціації: з боку "Користувача" 1, а з боку "Навчального курсу" діапазон кратності "0 .. 10" (від 0 до 10).

Це можна зробити двома способами:

l за допомогою спливаючого меню асоціації - вибрати в пункті Multiplicity (дослівно "Складність відносини") потрібну кратність;



l в діалоговому вікні властивостей асоціації (Open Specification - ) на вкладках RoleA Detail і RoleB Detail вибрати кратності для обох сторін асоціації зі списку пункту Multiplicity.



Отриманий взаємозв'язок (відношення):



Створимо аналогічно взаємозв'язок між класами "Студент" та "Навчальний курс":



Додавання на діаграму відносини узагальнення між двома класами виконується наступним чином. На спеціальній панелі інструментів необхідно натиснути кнопку із зображенням піктограми узагальнення і відпустити ліву кнопку миші. Далі на діаграмі класів треба виділити перший елемент узагальнення або нащадок, від якого виходить стрілка, і, не відпускаючи ліву кнопку миші, перемістити її покажчик до другого елементу відносини або предку, до якого спрямована стрілка.



Додати на діаграму відношення агрегації між двома класами можна наступними способами:

* Клацнути на кнопці з зображенням ставлення агрегації на спеціальній панелі інструментів і провести лінію агрегації від одного класу до іншого.
* Провести лінію асоціацію між вибраними класами і змінити її властивості таким чином, щоб перетворити дану асоціацію в агрегацію.

У другому випадку слід відкрити вікно специфікації властивостей асоціації Association Specification і на вкладці деталей відповідного кінця асоціації виставити відмітку в рядку вибору Aggregate (Агрегація).

Завдання

Побудувати за допомогою програми Rational Rose UML діаграму класів згідно варіанту.

Варіанти

**Варіанти**

**Варіант 1**

/\*

Super class - Page

\*/

class Page

{

private double width;

private double height;

//Constructor with no parameters

public Page()

{

width = 0;

height = 0;

}

//Constructor with paramaters

public Page(float width, float height)

{

this.width = width;

this.height = height;

}

//Destructor

~Page()

{

Console.WriteLine("Destructor");

}

//Method setting width

public void setWidth(float width)

{

this.width = width;

}

//Method getting width

public double getWidth()

{

return width;

}

//Method setting height

public void setHeight(float height)

{

this.height = height;

}

//Method getting height

public double getHeight()

{

return height;

}

//Static function

public static void staticFunction()

{

Console.WriteLine("Static function was called");

}

}

/\*

Class - Image

\*/

class Image

{

private float width;

private float height;

private String source;

//Constructor with paramaters

public Image(float width, float height, String source)

{

this.width = width;

this.height = height;

this.source = source;

}

//Method setting source

public void setSource(String source)

{

this.source = source;

}

//Method getting source

public String getSource()

{

return source;

}

}

/\*

Sub class - Main

\*/

class Main:Page

{

private static int count = 0;

private Image image = new Image(250, 250, "MainImage.jpg");

private String title;

private String text;

//Constructor with parameters

public Main(String title, String text)

{

count++;

this.title = title;

this.text = text;

Console.WriteLine("Creating "+ count + " Main Page");

}

//Method for changing image source

public void changeImageSource(String source)

{

image.setSource(source);

}

//Method getting source

public void getImageSource()

{

Console.WriteLine("Image source is: "+ image.getSource());

}

//Method getting count

public int getCount()

{

return count;

}

}

/\*

Sub class - About

\*/

class About : Page

{

private static int count = 0;

private Image image = new Image(250, 250, "AboutImage.jpg");

private String title;

private String text;

//Constructor with parameters

public About(String title, String text)

{

count++;

this.title = title;

this.text = text;

Console.WriteLine("Creating " + count + " About Page");

}

//Method for changing image source

public void changeImageSource(String source)

{

image.setSource(source);

}

//Method getting source

public void getImageSource()

{

Console.WriteLine("Image source is: " + image.getSource());

}

//Method getting count

public int getCount()

{

return count;

}

}

/\*

Sub class - News

\*/

class News : Page

{

private static int count = 0;

private Image image = new Image(250, 250, "NewsImage.jpg");

private String title;

private String text;

//Constructor with parameters

public News(String title, String text)

{

count++;

this.title = title;

this.text = text;

Console.WriteLine("Creating " + count + " News Page");

}

//Method for changing image source

public void changeImageSource(String source)

{

image.setSource(source);

}

//Method getting source

public void getImageSource()

{

Console.WriteLine("Image source is: " + image.getSource());

}

//Method getting count

public int getCount()

{

return count;

}

}

class Program

{

static void Main(string[] args)

{

//Caling static function

Page.staticFunction();

Console.WriteLine();

//Creating first Main page and changing all possible parameters

Main main1 = new Main("Hello", "This text references to Hello Main Page");

//Changing parameters

main1.changeImageSource("Hello.jpg");

main1.setHeight(400);

main1.setWidth(400);

//Displaying parameters

main1.getImageSource();

Console.WriteLine("Image width is: " + main1.getWidth());

Console.WriteLine("Image height is: " + main1.getHeight());

Console.WriteLine();

//Creating second Main page but leaving all parameters default

Main main2 = new Main("Goodbye", "This text references to Goodbye Main Page");

//Displaying parameters

main2.getImageSource();

Console.WriteLine("Image width is: " + main2.getWidth());

Console.WriteLine("Image heightf is: " + main2.getHeight());

Console.WriteLine();

//Displaying how many Main pages are created

Console.WriteLine("There are "+main1.getCount()+" Main pages created");

Console.ReadKey();

}

}

**Варіант 2**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba2

{

class Car

{

protected float weight;

protected float fuel;

protected float maxfuel;

public Car() { }

public Car(float w, float f, float max)

{

this.weight = w; this.fuel = f; this.maxfuel = max;

}

~Car()

{

Console.WriteLine("This is destructor of Car");

}

public void Go()

{

Console.WriteLine("Going");

}

public void Stop()

{

Console.WriteLine("Stoped");

}

public void DispIayParam()

{

Console.WriteLine("Weight=" + Convert.ToString(weight) + " Fuel=" + Convert.ToString(fuel) + " MaxFuel=" + Convert.ToString(maxfuel));

}

public static void Static()

{

Console.WriteLine("This static method");

}

}

class Truck:Car

{

private float cargo;

private float maxcargo;

private static int count = 0;

public Truck() { count++; }

public Truck(float w, float f, float max,float c,float mc):base(w,f,max)

{

cargo = c; maxcargo = mc; count++;

}

public void getCount()

{

Console.WriteLine("Count="+count);

}

public void DisplayParam()

{

Console.WriteLine("Cargo="+cargo+" MaxCargo="+maxcargo);

}

}

class SportCar : Car

{

private int count = 0;

private int maxspeed;

public SportCar() { count++; }

public SportCar(float w, float f, float max, int sp)

: base(w, f, max)

{

maxspeed = sp; count++;

}

public void DisplayMaxSpeed()

{

Console.WriteLine("MaxSpeed=" + Convert.ToString(maxspeed));

}

public void getCount()

{

Console.WriteLine("Count=" + count);

}

}

class Suv:Car

{

private float enginePower;

private static int count=0;

public Suv() { count++; }

public Suv(float w, float f, float max, float ep):base(w,f,max)

{ enginePower = ep;}

public void getCount()

{ Console. WriteLine("Count=" + count);}

public void displayPower()

{

Console. WriteLine("EnginePower="+Convert.ToString(enginePower));

}

}

class SmallCar: Car

{

private int count = 0;

private float power;

public SmallCar() { count++; }

public SmallCar(float w, float f, float max, float p) : base(w, f, max)

{power = p; count++;}

public void displayPower()

{Console. WriteLine("EnginePower=" + Convert.ToString(power));}

public void getCount()

{

Console.WriteLine("Count=" + count);

}

}

class Program

{

static void Main(string[] args)

{

Car c = new Car(200, 10, 20);

Truck t = new Truck(2000, 100, 150, 1000,5000);

SportCar sp = new SportCar(200, 50, 80, 300);

Suv s = new Suv(500, 100, 20, 800);

SmallCar sc = new SmallCar(150, 50, 80, 100);

sc.getCount();

t.DisplayParam();

c.Go();

c.Stop();

Car.Static();

t.DisplayParam();

Console.ReadKey();

}

}

}

**Варіант 3**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba2

{

class Building

{

protected float height;

protected float width;

protected float length;

public Building() { }

public Building(float h, float w, float l)

{ this.height = h;

this.width = w;

this.length = l;

}

public float Volume() {

return this.width \* this.length \* this.height;

}

~Building()

{

Console.WriteLine("Destructor");

}

public static void SMethod()

{

Console.WriteLine("Static Method");

}

}

class Garage : Building

{

private int numcar;

private static int count=0;

public Garage() { count++; }

public Garage(float h, float w, float l, int num): base(h, w, 1)

{

this.numcar = num; count++;

}

public void AddCar(int n)

{

this.numcar += n;

}

public int GetCar()

{

return this.numcar;

}

public void DisplayCount()

{ Console. WriteLine("Count= " + Convert. ToString(count));

}

}

class House : Building

{

private int numpeople;

private static int count=0;

public House() { count++; }

public House(float h, float w,float l, int n) : base(h, w, l)

{

this.numpeople = n; count++;

}

public void Display()

{

Console. WriteLine("Num of people "+Convert.ToString(this.numpeople)); }

public void DisplayCount()

{

Console.WriteLine("Count= "+Convert.ToString(count));

}

}

class Skyscrapen:Building

{

private static int count=0;

private int numfloor;

public Skyscrapen() { count++;}

public Skyscrapen(float h, float w,float l, int n) : base(h, w, l)

{ this.numfloor = n;

count++;

}

public void DisplayCount()

{

Console.WriteLine("Count=" + Convert.ToString(count));

}

public void Display()

{

Console. WriteLine("Num of floors " + Convert.ToString(this.numfloor));

}

}

class Theatre: Building

{

private static int count = 0;

private int numseats;

public Theatre() { count++;}

public Theatre(float h, float w, float l, int n): base(h, w, l)

{

this.numseats = n;

count++;

}

public void DisplayCount()

{

Console.WriteLine("Count=" + Convert.ToString(count));

}

public void Display()

{

Console.WriteLine("Num of seats " + Convert.ToString(this.numseats));

}

}

class Program

{

static void Main(string[] args)

{

Building.SMethod();

House h = new House(10, 20, 3, 5);

Garage g = new Garage(5, 5, 2, 2);

Skyscrapen s = new Skyscrapen(50, 50, 40, 15);

s.Display(); g.DisplayCount();

Console.WriteLine("Volume of house: " + Convert.ToString(h.Volume())); Building b = new Building(); Console.ReadKey();

}

}

}

**Варіант 4**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba2

{

class Figure

{

~Figure()

{

Console.WriteLine("Destructor");

}

public void displayDimensions()

{

Console.WriteLine("displayDimensions() from Figre public void Draw");

Console.WriteLine("Draw Figure");

}

public static void SMethod(){

Console.WriteLine("This is static method");

}

}

class Cone: Figure

{

private float r,h; // paAiyc OCHOBH i BMcora

public Cone(float r, float h)

{

this.r = r;

this.h = h; }

public void displayDimensions()

{

Console.WriteLine("r= " + r + " h= " + h);

}

public void Draw()

{

Console.WriteLine("Draw Cone");

}

}

class Pyramid : Figure

{

private float a, b, c, h;//cTopoHn OCHQBH i ewcoTa

public Pyramid(float a, float b, float c, float h)

{

this.a = a;

this.b = b;

this.c = c;

this.h = h;

}

public void displayDimensions()

{

Console.WriteLine("a=" + a + "b=" + b + "c=" + c + "h=");

}

public void Draw()

{

Console.WriteLine("Draw Pyramide");

}

}

class Bail: Figure

{

private float r;

public Bail(float r)

{

this.r = r;

}

public void displayDimensions()

{

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

}

public void Draw()

{

Console.WriteLine("Draw Ball");

}

}

class Program

{

static void Main(string[] args)

{

Cone c = new Cone(10f,20f);

Pyramid p = new Pyramid(1.2f, 3f, 2f, 5f);

Bail b = new Bail(4f);

Figure.SMethod();

Console.WriteLine("Cone dimensions");

c.displayDimensions();

Console.WriteLine("Ball dimensions");

b.displayDimensions();

b.Draw();

p.Draw();

Console.ReadKey();

}

}

}

**Варіант 5**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba2

{

class cTelefon

{

private String name;

private String model;

private String nommodel;

public cTelefon()

{

name = null; model = null; nommodel = null;

}

public cTelefon(string n, string m, string nom)

{

name = n; model = m; nommodel = nom;

}

~cTelefon()

{

Console.WriteLine("Destruktor");

}

public void Call(){

Console.WriteLine("Dzvonutu");

}

public void SendMassege()

{

Console.WriteLine("Send massege");

}

public void SetName(string n) {name = n;}

public string GetName(){ return name;}

public void SetModel(string m){

model = m;

}

public string GetModel(){ return model;}

public void SetModelNom(string nom) {nommodel = nom;}

public string GetModelNom() {return nommodel;}

public static void StaticFunction()

{

Console.WriteLine("cTelefon");

}

}

class Display

{

private String ModelDisplay;

private String SizeDisplay;

private int NumdersColor;

private bool Color;

public Display(string n,string size, int nom)

{

ModelDisplay = n;

SizeDisplay = size; NumdersColor = nom; Color = true;

}

public void IsColor()

{

Color = true;

}

public void IsWhiteBlack()

{

Color=false;

}

public bool getColor()

{

return Color;

}

}

class Operator

{

private int Codoper;

private String Nameoper;

public Operator(int id, string name)

{

Codoper = id; Nameoper = name;

}

public void setCodoper(int id) {Codoper = id;}

public int getCodoper() {return Codoper;}

public void setNameoper(string name){ Nameoper = name;}

public string getNameoper() {return Nameoper;}

}

class cMobilephone:cTelefon

{

private static int count = 0;

private int NumbersOfButtons;

private Display display =new Display("Saturn"," ",256);

private Operator operatorr = new Operator(8067, "Kyivstar ");

public cMobilephone(int n)

{

count++;

NumbersOfButtons = n; }

public int GetNumberOFButtons()

{

return NumbersOfButtons;

}

public int GetCount()

{

return count;

}

public void DisplayColor()

{

display.IsColor();

}

public void DisplayColorl()

{

display.IsWhiteBlack(); }

public bool getDisplayColor()

{

return display.getColor();

}

public string Nameoperator()

{

return operatorr.getNameoper();

}

public int CodOperatora ()

{

return operatorr.getCodoper();

}

}

class cStacionarphone : cTelefon

{

private static int count = 0;

private int NumbersOfButtons;

private Display display = new Display("Samsung", "240\*329", 1000000);

private Operator operatorr = new Operator(8063, "Life:)");

public cStacionarphone(int n){

count++; NumbersOfButtons = n;}

public int GetNumberOFButtons(){ return NumbersOfButtons;}

public int GetCount(){ return count;}

public void DisplayColor() {display.IsColor();}

public bool getDisplayColorl() {return display.getColor();}

public string Nameoperator()

{

return operatorr.getNameoper();

}

public int CodOperatora()

{

return operatorr.getCodoper();

}

}

class Program

{

static void Main(string[] args)

{

cTelefon.StaticFunction();

cMobilephone mobilel=new cMobilephone(16);

cMobilephone mobile2 = new cMobilephone(12); cMobilephone mobile3 = new cMobilephone(15); cMobilephone mobile4 = new cMobilephone(20); cStacionarphone statcphonel = new cStacionarphone(9); cStacionarphone statcphone2 = new cStacionarphone(12);

mobilel.Call(); mobilel.SendMassege();

Console.WriteLine("Count mobile: " + mobilel.GetCount());

Console.WriteLine("Color display in mobilel " + mobilel.getDisplayColor());

mobile2.DisplayColor();

Console.WriteLine("Color display in mobile 2: " + mobile2.getDisplayColor());

mobile2.DisplayColorl();

Console.WriteLine("Color display in mobile 2: " + mobile2.getDisplayColor());

Console.WriteLine("Numbers of Buttons: "+mobilel.GetNumberOFButtons());

Console.WriteLine("Count stacionarphone: " + statcphone2.GetCount());

Console.WriteLine("Name oper-" + mobilel.Nameoperator());

Console.WriteLine("Cod oper: " + mobilel.CodOperatora());

Console.WriteLine("Name oper-" + statcphonel.Nameoperator());

Console.WriteLine("Cod oper: " + statcphonel.CodOperatora());

Console.ReadKey();

}

}

}