**Лабораторна робота №6 Технологія створення програмних продуктів з використанням UML**

**Мета:** **Навчитися будувати діаграми** **компонентів та розгортання**

**Завдання до лабораторної роботи**

1. Ознайомитись з теоретичною частиною та рекомендаціями до виконання роботи.
2. Запустити Umbrello UML Modeller (в разі відсутності роботу виконувати з іншим доступним засобом UML-моделювання)
3. Відкрити попередній проект та за аналогією до розгянутого в теоретичній частині прикладу додати до розробленої діаграми класів клас IdCard (Перепустка), Room (Приміщення), Faculty (Факультет) (по аналогії з відділом), PastPosition (Попередня посада) з аналогічними властивостями та методами. Встановити за аналогією з прикладом зв’язки між об’єктами. Згенеруйте текст на С++ та визначте в який мірі його потрібно допрацювати для конкретного використання та яку функціональність та описи до нього додати. Викласти письмово.
4. До діаграм додати нотатки, де записати своє прізвище та групу, а також номер ЛР.
5. Для відсилки викладачу збережіть проект з іменем Вашої ЛР та розширенням xmi.
6. Оформите звіт, в якому надайте скріншоти створеної діаграми та відповіді на запитання в кінці ЛР.
7. Файл зі звітом викласти на платформу коледжу до репозиторію викладача вашого предмету з іменем у форматі :

**PI<Номер групи><Номер лекції / практичної / лабораторної>[-<Номер завдання>][літера позначення типу роботи L – лекція, P –практична, R – лабораторна]<Прізвищеанглійською>**. Наприклад, **PI3103R**buts.doc.

1. Файл зі звітом та створену Umbrello UML Modeller модель (файл xmi) відішліть на електронну адресу викладача.
2. **Строк виконання цієї роботи ІПЗ-31 07.05.2020, ІПЗ-32 27.04.2020**

**ТЕОРЕТИЧНА ЧАСТИНА**

Діаграма класів UML дозволяє позначати відношення між класами та їх екземплярами, в чому зазвичай виникає потреба для моделювання прикладної області. Розглянемо як ці відношення відображуються у програмному коді. Спочатку розглянемо відношення між класами в UML. Надана на рис.1 структурна схема демонструє різновиди відношень.

Асоціації мають навігацію: двонаправлену або однонаправлену, яка вказує на напрямок зв’язку. Тобто у кожного виду асоціації ще є два підвиди, які на рисунку не показані.

***1. Узагальнення***

Побудуємо UML-діаграму класів (Class Model), а потім відобразимо її в об’єктно-орієнтованому коді. В якості прикладної області візьмемо відділ кадрів деякого підприємства та побудуємо його модель. Для прикладів будемо використовувати мову Java.

Відношення узагальнення — це спадкування. В мові Java воно має явну реалізацію через розширення(extends) одного класу іншим.

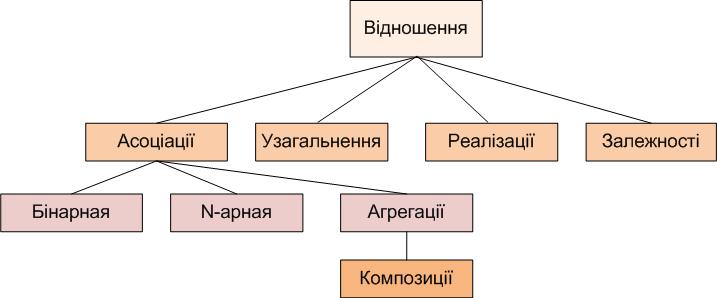


Рисунок 1 Відношення між класами

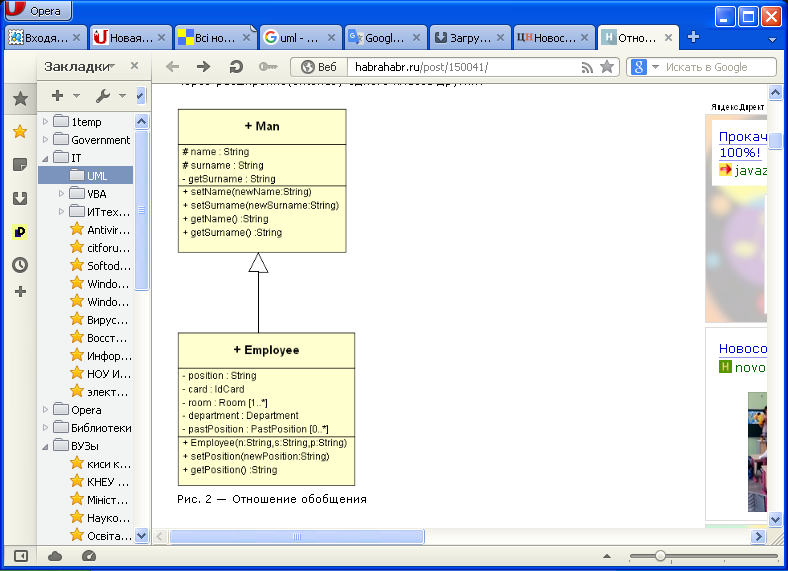


Рисунок 2. Відношення узагальнення

Клас «Man»(людина) — більш абстрактний, а «Employee»(співробітник) більш спеціалізований. Клас «Employee» успадковує властивості та методи «Man».

Код для цієї діаграми:

public class Man{

protected String name;

protected String surname;

public void setName(String newName){

name = newName;

}

public String getName(){

return name;

}

public void setSurname(String newSurname){

name = newSurname;

}

public String getSurname(){

return surname;

}

}

// успадковуємо клас Man

public class Employee extends Man{

private String position;

// створюємо конструктор

public Employee(String n, String s, String p){

name = n;

surname = s;

position = p;

}

public void setPosition(String newProfession){

position = newProfession;

}

public String getPosition(){

return position;

}

}

***2. Асоціація***

Асоціація показывает отношения между объектами-экземплярами класса.

***2.1 Бінарна***

В модель добавили клас «IdCard», який представляє ідентифікаційну картку(пропуск) співробітника. Кожному співробітнику може відповідати тільки одна ідентифікаційна картка, множинність зв’язку 1 до 1.

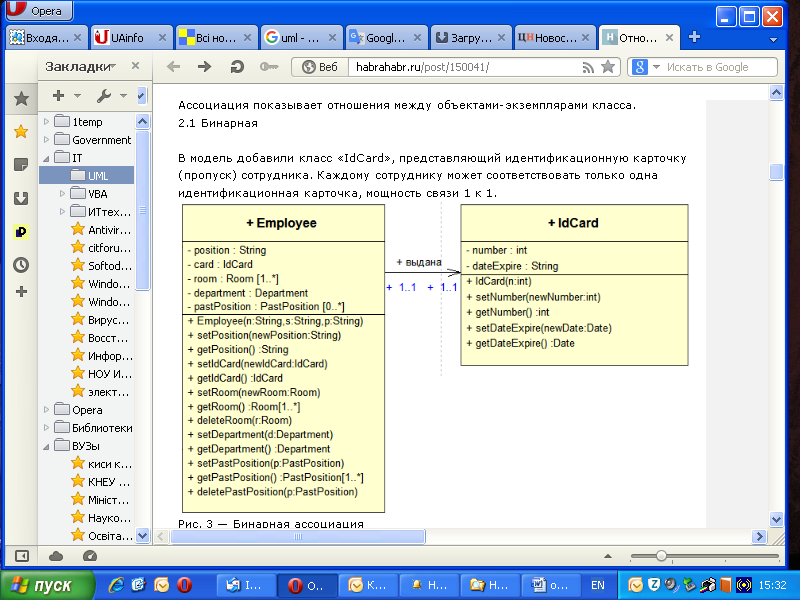


Рисунок 3 — Бінарна асоціація

Класи:

public class Employee extends Man{

private String position;

private IdCard iCard;

public Employee(String n, String s, String p){

name = n;

surname = s;

position = p;

}

public void setPosition(String newPosition){

position = newPosition;

}

public String getPosition(){

return position;

}

public void setIdCard(IdCard c){

iCard = c;

}

public IdCard getIdCard(){

return iCard;

}

}

public class IdCard{

private Date dateExpire;

private int number;

public IdCard(int n){

number = n;

}

public void setNumber(int newNumber){

number = newNumber;

}

public int getNumber(){

return number;

}

public void setDateExpire(Date newDateExpire){

dateExpire = newDateExpire;

}

public Date getDateExpire(){

return dateExpire;

}

}

В тілі програми створюємо об’єкти та зв’язуємо їх:

IdCard card = new IdCard(123);

card.setDateExpire(new SimpleDateFormat("yyyy-MM-dd").parse("2015-12-31"));

sysEngineer.setIdCard(card);

System.out.println(sysEngineer.getName() +" works in a post "+ sysEngineer.getPosition());

System.out.println("The certificate is valid until " + new SimpleDateFormat("yyyy-MM-dd").format(sysEngineer.getIdCard().getDateExpire()) );

Класс Employee має поле card, у якого тип IdCard, також клас має методи для присвоювання значення (setIdCard) цьому полю и для отримання значення (getIdCard). З екземпляру об’єкта Employee можемо узнати про зв’язаний з ним об’єкт типу IdCard. Це означає, що навігація (стрілка на лінії) направлена від Employee до IdCard.

***2.2 N-арна асоціація***

Нехай в організації положено закріплювати за робітниками приміщення. Додано новий клас Room.

Кожному об’єкту робітник(Employee) може відповідати декілька робочих приміщень. множинність зв’язку один-до-багатьох.

Навігація від Employee до Room.

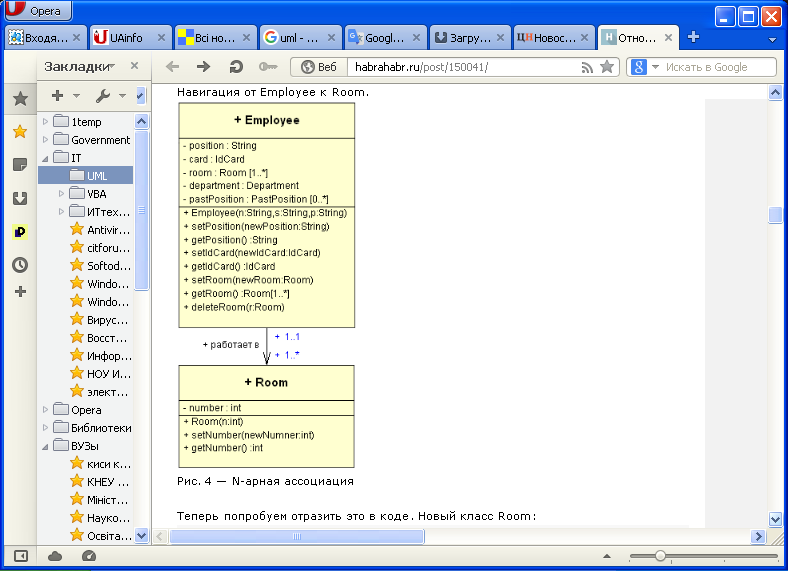


Рисунок 4 — N-арна асоціація

Це таким чином відображується в коді. Новий клас Room:

public class Room{

private int number;

public Room(int n){

number = n;

}

public void setNumber(int newNumber){

number = newNumber;

}

public int getNumber(){

return number;

}

}

Додаємо в клас Employee поле та методи для роботи з Room:

...

private Set room = new HashSet();

...

public void setRoom(Room newRoom){

room.add(newRoom);

}

public Set getRoom(){

return room;

}

public void deleteRoom(Room r){

room.remove(r);

}

...

Приклад використання:

public static void main(String[] args){

Employee sysEngineer = new Employee("John", "Connor", "Manager");

IdCard card = new IdCard(123);

card.setDateExpire(new SimpleDateFormat("yyyy-MM-dd").parse("2015-12-31"));

sysEngineer.setIdCard(card);

Room room101 = new Room(101);

Room room321 = new Room(321);

sysEngineer.setRoom(room101);

sysEngineer.setRoom(room321);

System.out.println(sysEngineer.getName() +" works in a post "+ sysEngineer.getPosition());

System.out.println("The certificate is valid until " + sysEngineer.getIdCard().getDateExpire());

System.out.println("It can be indoors:");

Iterator iter = sysEngineer.getRoom().iterator();

while(iter.hasNext()){

System.out.println( ((Room) iter.next()).getNumber());

}

}

***2.3 Агрегація***

Введемо до моделі клас Department(відділ) —підприємство, що ми розглядаємо, структуровано по відділах. В кожному відділі може працювати один або більше людей, тобто відділ включає в себе одного або більше співробітників і таким чином їх агрегує. На підприємстві можуть бути співробітники, які не належать до жодного відділу, наприклад, директор підприємства.

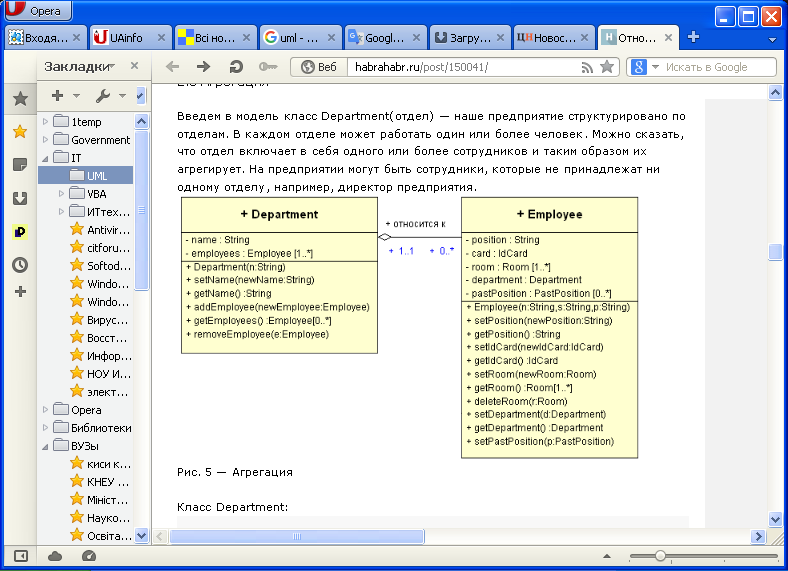


Рисунок 5. Агрегація

Клас Department:

public class Department{

private String name;

private Set employees = new HashSet();

public Department(String n){

name = n;

}

public void setName(String newName){

name = newName;

}

public String getName(){

return name;

}

public void addEmployee(Employee newEmployee){

employees.add(newEmployee);

// зв’язуємо співробітника з цим відділом

newEmployee.setDepartment(this);

}

public Set getEmployees(){

return employees;

}

public void removeEmployee(Employee e){

employees.remove(e);

}

}

Таким чином наш клас, окрім конструктора та методу зміни імені відділу, має методи для занесення до відділу нового співробітника, для видалення співробітника та для отримання всіх співробітників, що входять до цього відділу. Навігація на діаграмі не показана, що означає, що вона є двонаправленою: від об’єкту типу «Department» можна узнати про співробітника та від об’єкта типу «Employee» можна узнати до якого відділу він відноситься.

Оскільки нам потрібно легко узнавати, до якого відділу відноситься будь-який співробітник, то додаємо до класу Employee поле та методи для призначення та отримання відділу.

...

private Department department;

...

public void setDepartment(Department d){

department = d;

}

public Department getDepartment(){

return department;

}

Використання:

Department programmersDepartment = new Department("Programmers ");

programmersDepartment.addEmployee(sysEngineer);

System.out.println("Refers to the department "+sysEngineer.getDepartment().getName());

***2.3.1 Композиція***

Нехай однією з вимог до нашої системи є вимога зберігання даних про попередню посаду, яку займав співробітник на підприємстві.

Введемо новий клас «pastPosition». До нього окрім властивості «ім’я»(name), введемо ще властивість «department», яка зв’яже його з класом «Department».

Дані про попередні посади, які займав співробітник, є частиною даних про співробітника, таким чином між ними зв’язок ціле-частина та в той же час, дані про попередні посади не можуть існувати без об’єкта типу «Employee». Знищення екземпляру об’єкту «Employee» повинно привести до знищення відповідних екземплярів об’єктів «pastPosition».

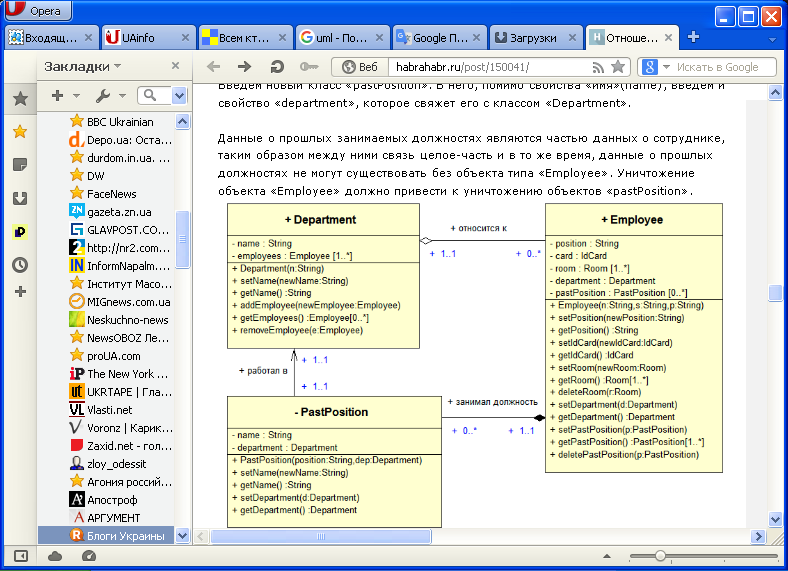


Рисунок 6. Композиція

Клас «PastPosition»:

private class PastPosition{

private String name;

private Department department;

public PastPosition(String position, Department dep){

name = position;

department = dep;

}

public void setName(String newName){

name = newName;

}

public String getName(){

return name;

}

public void setDepartment(Department d){

department = d;

}

public Department getDepartment(){

return department;

}

}

До класу Employee додаємо властивості та методи для роботи з даними щодо попередньої посади:

...

private Set pastPosition = new HashSet();

...

public void setPastPosition(PastPosition p){

pastPosition.add(p);

}

public Set getPastPosition(){

return pastPosition;

}

public void deletePastPosition(PastPosition p){

pastPosition.remove(p);

}

...

Застосування:

// змінюємо посаду

sysEngineer.setPosition("Watchman ");

// дивимось на раніш займаємі посади:

System.out.println("Worked in the past as:");

Iterator iter = sysEngineer.getPastPosition().iterator();

while(iter.hasNext()){

System.out.println( ((PastPosition) iter.next()).getName());

}

***3. Залежність***

Для організації діалогу з користувачем введемо до системи клас «Menu». Вбудуємо один метод «showEmployees», який показує список співробітників та їх посади. Параметром для методу є масив екземплярів об’єкту «Employee». Таким чином, внесені до класу «Employee» зміни можуть вимагати й змін класу «Menu».

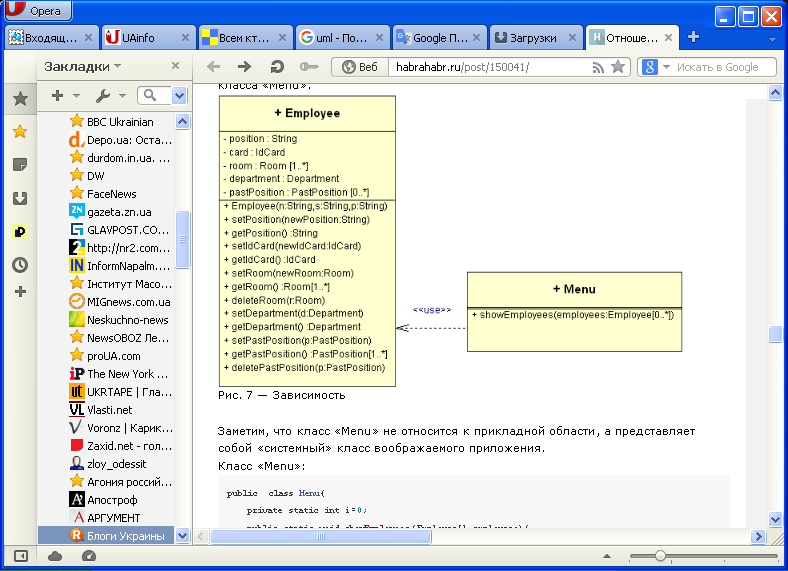


Рисунок 7 — Залежність

Клас «Menu» не відноситься до прикладної області, а представляє собою «системний» клас планованого застосунку.

Клас «Menu»:

public class Menu{

private static int i=0;

public static void showEmployees(Employee[] employees){

System.out.println("A list of employees:");

for (i=0; i<employees.length; i++){

if(employees[i] instanceof Employee){

System.out.println(employees[i].getName() +" - " + employees[i].getPosition());

}

}

}

}

Використання:

// додамоеще одного співробітника

Employee director = new Employee("Alex", "Ivanov", " Director ");

Menu menu = new Menu();

Employee employees[] = new Employee[10];

employees[0]= sysEngineer;

employees[1] = director;

Menu.showEmployees(employees);

***4. Реалізація***

Реалізація, як і спадкування має явний вираз в мові Java: оголошення інтерфейсу та можливість його реалізації будь-яким класом.

Для демонстрації відношення «реалізація» створимо інтерфейс «Unit». Якщо представити, що організація може ділитися не тільки на відділи, але й наприклад, на філіали тощо. Інтерфейс «Unit» представляє собою саму абстрактну одиницю ділення. В кожній одиниці ділення працює деяка кількість співробітників, тому метод для отримання кількості працюючих людей буде актуальним для кожного класу, який реалізує інтерфейс «Unit».

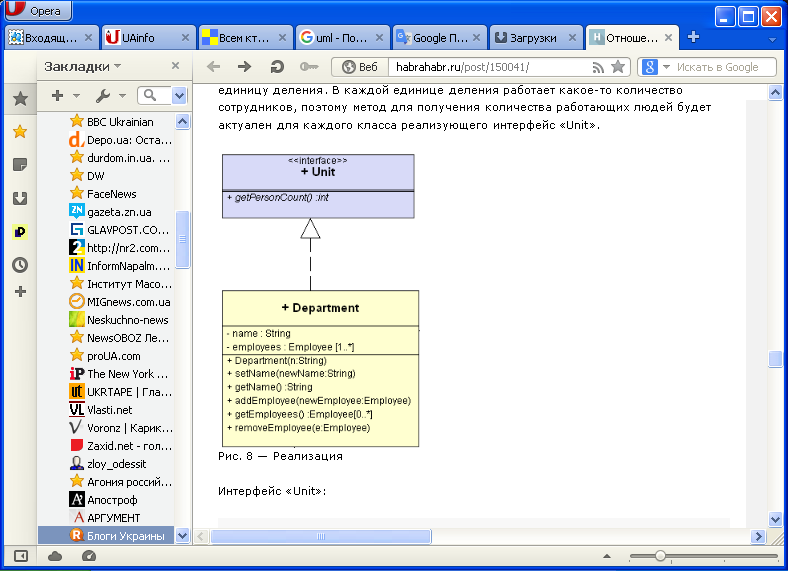


Рисунок 8. Реалізация

Інтерфейс «Unit»:

public interface Unit{

int getPersonCount();

}

Реалізація в класі «Department»:

public class Department implements Unit{

...

public int getPersonCount(){

return getEmployees().size();

}

Застосування:

System.out.println("В отделе "+sysEngineer.getDepartment().getName()+" work "

+sysEngineer.getDepartment().getPersonCount()+" person.");

Як бачимо, реалізація методу «getPersonCount» не зовсім актуальна для класу «Department», оскільки він має метод «getEmployees», який повертає колекцію екземплярів об’єкту «Employee».

Код повністю: http://code.google.com/p/umljava/downloads/list

***Висновки***

Мова моделювання UML має набір відносин для побудови моделі класів, але навіть така розвинута ООП мова, як Java має тільки дві явні конструкції для відображення зв’язків: extends(розширення) и interface/implements(реалізація).

В результате моделювання отримали таку діаграму:

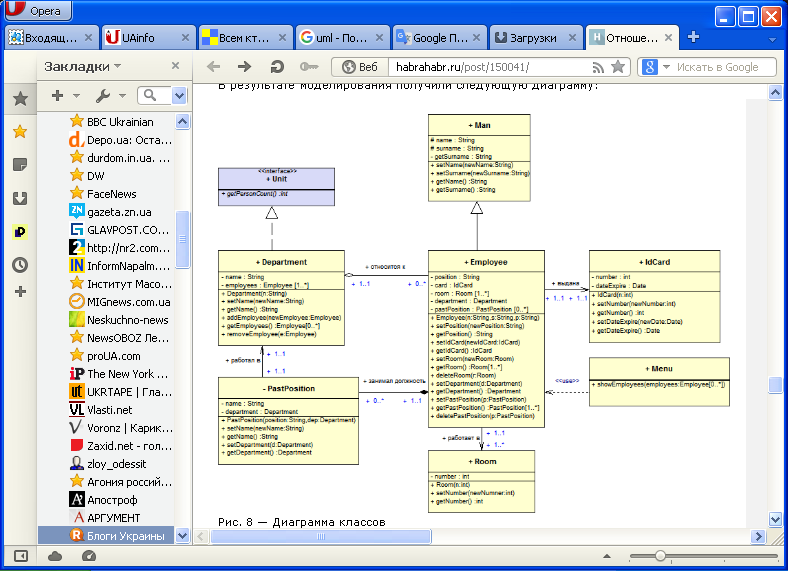


Рисунок 8 — Діаграма класів

**Литература**

1) Г. Буч, Д. Рамбо, А. Джекобсон. Язык UML Руководство пользователя.

2) А.В. Леоненков. Самоучитель UML

3) Эккель Б. Философия Java. Библиотека программиста. — СПб: Питер, 2001. — 880 с.

4) Орлов С. Технологии разработки программного обеспечения: Учебник. — СПб: Питер, 2002. — 464 с.

5) Мухортов В.В., Рылов В.Ю.Объектно-ориентированное программирование, анализ и дизайн. Методическое пособие. — Новосибирск, 2002.

6) Anand Ganesan. Modeling Class Relationships in UML

**Контрольні запитання**.

1. Чи може діаграма класів показати динамічну поведінку системи?
2. Визначте основні будівельні блоки UML?
3. Які цілі UML?
4. Яка роль UML в ОО-аналізе та проектуванні?