Information Systems Engineering and Management 4

S. Manoharan
Alexandru Tugui
Zubair Baig *Editors*

# Proceedings of 4th International Conference on Artificial Intelligence and Smart Energy

## ICAIS 2024, Volume 2

Springer

# Information Systems Engineering and Management 4

The book series "Information Systems Engineering and Management" (ISEM) publishes innovative and original works in the various areas of planning, development, implementation, and management of information systems and technologies by enterprises, citizens, and society for the improvement of the socio-economic environment.

The series is multidisciplinary, focusing on technological, organizational, and social domains of information systems engineering and management. Manuscripts published in this book series focus on relevant problems and research in the planning, analysis, design, implementation, exploration, and management of all types of information systems and technologies. The series contains monographs, lecture notes, edited volumes, pedagogical and technical books as well as proceedings volumes.

Some topics/keywords to be considered in the ISEM book series are, but not limited to: Information Systems Planning; Information Systems Development; Exploration of Information Systems; Management of Information Systems; Blockchain Technology; Cloud Computing; Artificial Intelligence (AI) and Machine Learning; Big Data Analytics; Multimedia Systems; Computer Networks, Mobility and Pervasive Systems; IT Security, Ethics and Privacy; Cybersecurity; Digital Platforms and Services; Requirements Engineering; Software Engineering; Process and Knowledge Engineering; Security and Privacy Engineering, Autonomous Robotics; Human-Computer Interaction; Marketing and Information; Tourism and Information; Finance and Value; Decisions and Risk; Innovation and Projects; Strategy and People.

Indexed by Google Scholar. All books published in the series are submitted for consideration in the Web of Science.

For book or proceedings proposals please contact Alvaro Rocha (amrrocha@gmail.com).

S. Manoharan · Alexandru Tugui · Zubair Baig
Editors

# Proceedings of 4th International Conference on Artificial Intelligence and Smart Energy

## ICAIS 2024, Volume 2

Springer

*Editors*
S. Manoharan
JCT College of Engineering and Technology
Coimbatore, Tamil Nadu, India

Zubair Baig
School of Information Technology
Deakin University
Geelong, Australia

Alexandru Tugui
Department of Business Informatics
Alexandru Ioan Cuza University
Iasi Romania, Romania

ISSN 3004-958X ISSN 3004-9598 (electronic)
Information Systems Engineering and Management
ISBN 978-3-031-61474-3 ISBN 978-3-031-61475-0 (eBook)
https://doi.org/10.1007/978-3-031-61475-0



This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*We would like to dedicate this proceeding to all members of the advisory committee and program committee for providing their excellent guidance. We also dedicate this proceeding to the members of the review committee for their excellent cooperation throughout the conference. Additionally, we extend our sincere thanks to all the authors and participants.*

# Preface

Welcome to the second volume of the book, Artificial Intelligence and Smart Energy, where innovative research and cutting-edge technologies converge to address the challenges of our time. This volume presents a comprehensive collection of research chapters from diverse domains, each contributing significantly to the advancement of energy and sustainability.

This book reflects the collective efforts of researchers, scholars, and practitioners from around the globe, showcasing their dedication to exploring novel solutions to complex problems. From securing cyber-physical systems to revolutionizing healthcare with artificial intelligence and robotics, the topics covered here highlight the ways to leverage sustainable research.

Moreover, this volume is particularly focused on smart and sustainable energy solutions, which plays a major role in building a greener future for generations to come. From advancements in solar panel efficiency and fault analysis to the development of fuzzy-controlled converters for grid-tied photovoltaic systems, the chapters presented in this volume offer insights into the latest innovations in renewable energy technologies. On the other hand, by including the research studies on water solutions, agricultural innovations, and renewable energy technologies, this volume depicts the transformative potential of artificial intelligence, machine learning, and the Internet of Things across various domains, including smart cities, transportation, and healthcare.

We believe that the insights shared in this volume will inspire further research, enable fruitful discussions, and create real-world impact to drive toward a future defined by resilience, inclusivity, and sustainable development.

S. Manoharan
Alexandru Tugui
Zubair Baig

# Contents

# Formation of LMS Class Diagram

Elov Botir Boltayevich[1], Toirova Guli Ibragimovna[2], Zuparov Talat Marufovich[1], and Axmedova Xolisxon Ilxomovna[1(✉)]

[1] Tashkent State University of Uzbek Language and Literature Named After Alisher Navo'i, Tashkent, Uzbekistan
{elov,zuparovtalat}@navoiy-uni.uz, xolisa9029@mail.ru
[2] Bukhara State University, Bukhara, Uzbekistan

**Abstract.** Class diagrams show the classes in an information system and the relationships between these classes. A class diagram is a Unified Modeling Language (UML) diagram widely used in education. Class diagrams are mainly important for the design and development of information systems. This article describes LMS (Learning Process Management System) objects using UML class diagrams, which are considered a modern approach to information systems design. Class diagrams show the attributes, methods, and relationships between LMS objects such as department, employee, training program, and training program periods. Also, a practical application of LMS class diagrams in C# language was formed.

**Keywords:** UML diagram · model · class diagram · inheritance · association · aggregation · dependency · implementation

## 1 Introduction

Based on the model-driven architecture (MDA), models and diagrams are used in the process of visualization and development of the company's information systems. Therefore, the BPMN (business process model and notations) method is used in the development of information systems. It is appropriate to describe the processes in it through diagrams so that all ordinary users of the organization's information systems can quickly and easily understand their needs and goals. Plat- form-independent models are formed in the information system developer by transferring BPMN diagrams to the Unified Modeling Language (UML) class diagram as an independent computational model [1, 2].

Class diagram (class diagram) is the main method of describing the static structure of the system. The relationship between classes and their copies is represented by a class diagram [3–5]. Class diagrams are used in the modeling of practical areas.

UML is the de facto standard notation for graphical representation of an information system. UML diagrams are used in the analysis, construction and maintenance of information systems. Often, UML diagrams take an abstract view of a software system. The main purpose of UML diagrams is to share knowledge about the system between programmers. The layout quality of UML diagrams plays a crucial role in understanding them.


S. Manoharan et al. (Eds.): ICAIS 2024, ISEM 4, pp. 493–509, 2024.
https://doi.org/10.1007/978-3-031-61475-0_39

## 2 A Review of Applied Research

In 1990s object-oriented languages such as C++ developed highly. These object-oriented languages have been used to create complex but highly efficient information systems. Since the developed information systems are difficult to understand, this has led to design and analysis problems faced after system implementation [4, 6, 7]. Therefore, the information system was very difficult to explain to others.

After the creation of the UML language, many experiments and approaches were implemented to simplify such difficult tasks of information system analysis [8]. UML is a unified object-oriented modeling language, which was proposed in 1994–1995 by Rational software company engineers Grady Booch, Ivar Jacobson and James Rumbaugh [9]. They put forward great ideas for designing a language that would reduce the complexity of the information system development process.

Booch method is very flexible to work in the design and construction of objects [10]. Jacobson presented a unique method of working on system states [11]. It should be noted that Jacobson's method has a strong approach to high-level design. The Rumbaugh method turned out to be very useful in working with complex systems [12]. Today, feature models and state diagrams, invented by David Harrell, are widely used in UML.

In 1997, UML was recognized as a standard by the Object Management Group (OMG) [13]. The Object Management Group is the organization responsible for managing UML since its adoption as a standard. In 2005, the International Organization for Standardization (ISO) approved UML as an ISO standard. Since then, UML has been used in various fields to create object-oriented models. The latest UML version is 2.5.1, developed and implemented in December 2017 [14].

## 3 Methodology

LMS has several component systems, which in turn embody several business processes [15, 16]. Fig. 1 below shows the class diagram of the business process objects of creating LMS educational programs and keeping professor-teacher accounts.

A class diagram uses one basic "class" for objects. In some cases, it is possible to use interfaces, primitive types, and associations that are special cases of classes. The following types of relationships can be established between these classes:

- inheritance;
- association;
- dependence;
- implementation.

Using data from different sources, a structure representing relationships can be formed as follows:

1. Inheritance

The relationship between the general element and the private element is established through inheritance.

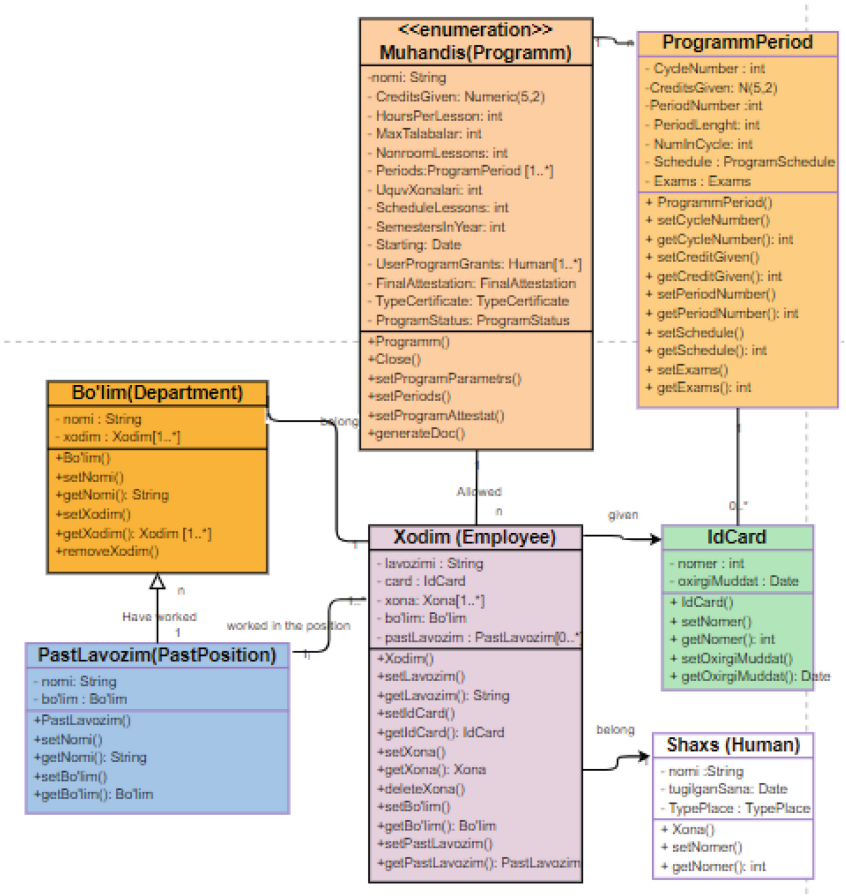


**Fig. 1.** Class diagram of the business process objects of creating educational programs and maintaining faculty accounts

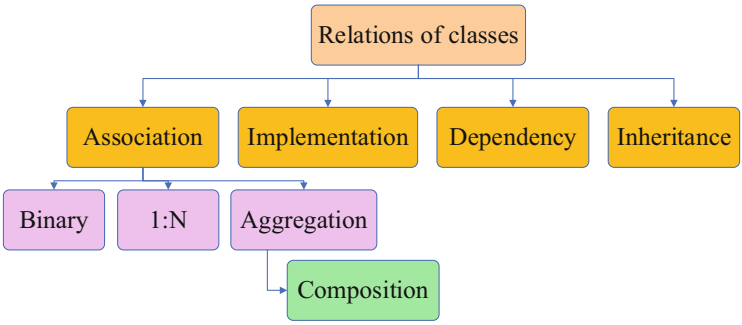


**Fig. 2.** Relations between classes

Relations between classes is shown in Fig. 2.This relationship is well covered in the resources on object-oriented programming languages. In C#, one class inherits from another class like `public class Xodim: Shaxs`

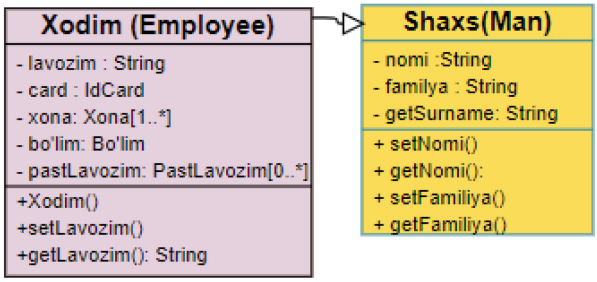


**Fig. 3.** "Inheritance" relation

The **«Shaxs»**class is abstract, and the **«Xodim»**class is an implementation. The **«Xodim»**class inherits the properties and methods of the **«Shaxs»**class. The program code corresponding to the diagram in Fig. 3 is as follows:

```
public class Shaxs{
 protected String ismi;
 protected String familiyasi;
 public void setIsm(String newIsm){
     ismi = newIsm;
 }
 public String getIsm(){
     return ismi;
 }
 public void setFam(String newFam) {
     familiyasi = newFam;
 }
 public String getFam(){
     return familiyasi;
 }
    }

  public class Xodim: Shaxs{
     private String lavozim;
     // constructor
     public Xodim(String n, String s, String p) {
         ismi = n;
         familiyasi = s;
         lavozim = p;
     }
     public void setLavozim(String newLavozim) {
         Lavozim = newLavozim;
     }
     public String getLavozim(){
         return Lavozim;  }   }
```

Classes and their properties and methods form a hierarchical structure in a class diagram formed by inheritance relationships. The successor class will have all the properties and methods of the source class. A resource can also have its own properties and methods that are not available in a class.

2. Association

By association, a long line is formed between the classes, and the description of the relationship is shown using special keywords.

Relationships can be formed in 3 different ways:

2.1. Binary;
2.2. 1:N;
2.3. Aggregation
2.3.1. ComLavozim

An association can have two directions: unidirectional and bidirectional. Navigation will show the direction of communication.

## 2.1 Binary association

The Xodim identification process is implemented by adding the "IdCard" class to the model. Only one identification card (IdCard) is suitable for each Xodim. The relationship between the two classes is set as 1:1. Binary association is shown in Fig. 4.

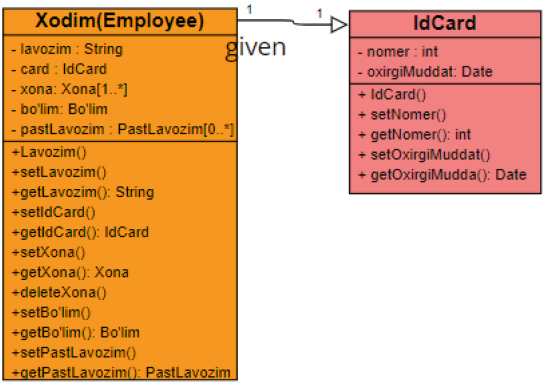


**Fig. 4.** Binary association

Classes:

```
public class Shaxs{
    protected String ismi;
    protected String familiyasi;
    public void setIsm(String newIsm){
        ismi = newIsm;
    }
    public String getIsm(){
            return ismi;
        }
     public void setFamiliyasi(String newFamiliyasi){
            familiyasi = newFamiliyasi;
        }
     public String getFamiliyasi(){
            return familiyasi;
        } }
```

```
        public class Xodim : Shaxs {
            private String Lavozim;
            private IdCard idCard;
            public Xodim(String n, String s, String p){
                ismi = n;
                familiyasi = s;
                Lavozim = p;
            }
            public void setLavozim(String newLavozim){
                lavozim = newLavozim;
            }
            public String getLavozim(){
                return lavozim;
            }
            public void setIdCard(IdCard c){
                iCard = c;
            }
            public IdCard getIdCard(){
                return iCard;
            }
        }


public class IdCard
    {
        private DateTime OxirgiMuddat;
        private int nomer;
        public IdCard(int n){
            number = n;
        }
        public void setNomer (int newNomer) {
            nomer = newNomer;
        }
        public int getNomer(){
            return nomer;
        }
        public void setOxirgiMuddat(DateTime newOxirgiMuddat){
            oxirgiMuddat = newOxirgiMuddat;
        }
        public DateTime getOxirgiMuddat(){
            return OxirgiMuddat;
        } }
```

In the program code, we create objects and connect them:

```
static void Main(string[] args) {
    Xodim Muhandis = new Xodim("Hamraev","Otabek","Injener");
    IdCard card = new IdCard(123);
    card.setOxirgiMuddat(new DateTime(2023, 12, 31));
    Muhandis.setIdCard(card);
    Console.WriteLine(Muhandis.getIsm() +" : "+
Muhandis.getLavozim()+ " - lavozimida ishlaydi.");
     Console.WriteLine("IdCard " +
Muhandis.getIdCard().getOxirgiMuddat().ToString() + " muddatgacha
amal qiladi ");}
```

The Xodim class has a **card** attribute of the IdCard type, and has methods **setIdCard** that assign a value to this attribute and **getIdCard** that retrieves a value from the attribute. Because the **Muhandis** object copied from the Xodim class is linked to the IdCard object, the routing is from Xodim to IdCard.

### 2.2 1:N association

Xodimlar of the organization should be assigned to Xonas. For this, we will add the Xona class to the project. Each Xodim can have several jobs. Therefore, the relationship is formed as **1:n** (one-to-many) and the direction is from Xodim to Xona. 1:n association is shown in Fig. 5.

Now we will reflect the objects in this diagram to the program code. Add the new Xona class to the project:

```
public class Xona{
   private int nomer;
   public Xona(int n){
      nomer = n;}
   public void setNomer(int newNomer){
       nomer = newNomer;}
   public int getNomer(){
       return number;}}
```

We add attributes and methods for working with Xona to the Xodim class:

```
...
List<Xona> Xona= new List<Xona>();
public void setXona(Xona newXona){
   Xona.Add(newXona);}
public List<Xona> getXona(){
   return Xona;}
public void deleteXona(Xona r){
   Xona.Remove(r);}
...
```

An example of using these classes:

```
      static void Main(string[] args){
        Xodim Muhandis = new Xodim("Hamraev","Otabek","Manager");
      // Add IdCard
        IdCard card = new IdCard(123);
        card.setOxirgiMuddat(new DateTime(2023, 12, 31));
        Muhandis.setIdCard(card);
      // add Xonas
         Xona Xona101 = new Xona(101);
         Xona Xona321 = new Xona(321);
         Muhandis.setXona(Xona101);
         Muhandis.setXona(Xona321);
         Console.WriteLine(Muhandis.getIsm() +":"+
Muhandis.getLavozim()+ "- lavozimida ishlaydi.");
         Console.WriteLine("IdCard " +
Muhandis.getIdCard().getOxirgiMuddat().ToString() + "muddatgacha
amal qiladi ");
         Console.WriteLine("IdCard" +
Muhandis.getIdCard().getOxirgiMuddat().ToString() + "muddatgacha
amal qiladi ");
         Console.WriteLine("Xodim kuyidagi xonalarga kirishi
mumkin:");
         foreach (Xona Xona in Muhandis.getXona()){
                    Console.WriteLine(Xona.getNumber());}
             }
```

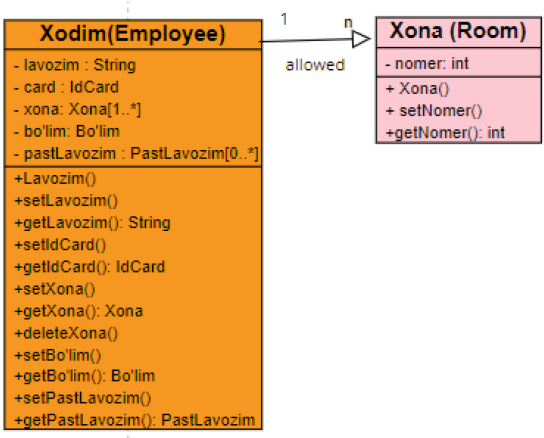


**Fig. 5.** 1:n association

## 2.3 Aggregation

We will add a new Bo`limclass to the project model. Each Bo‘lim has one or more Xodimlar. Since a Bo‘lim can hold one or more Xodimlar, it aggregates them (Fig. 6). Xodimlar who do not belong to any Bo‘lim (the director of the organization) can work in the organization. Aggregation relation is shown in Fig. 6.

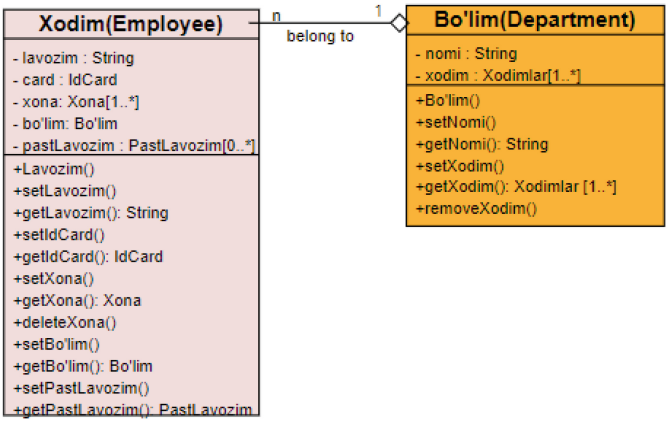


**Fig. 6.** Aggregation relation

```
public class Bo`lim{
   private String nomi;
   List<Xodim> Xodimlar = new List<Xodim>();
   public Bo`lim(String n) {
       nomi = n;       }
   public void setNom(String newNom) {
       nomi = newNom;
     }
   public String getNom(){
       return nomi;
   }
   public void addXodim(Xodim newXodim){
       Xodimlar.Add(newXodim);
// Attach an Xodim to a Bo`lim
       newXodim.setBo`lim(this);
   }
   public List<Xodim> getXodimlar(){
       return Xodimlar;
   }
   public void removeXodim(Xodim e){
                   Xodimlar.Remove(e)}}
```

The **Bo`lim** class contains methods such as constructor, adding a new Xodim to the Bo‘lim, deleting and getting, m…… information about the Xodimlar attached to the Bo‘lim. The direction is not shown in the diagram. So, **Bo`lim** and **Xodim** objects have a two-way relationship. Therefore, it is possible to determine about the Xodim(s) through the **Bo`lim** object, and in which Bo‘lim he/she works through the **Xodim** object. To perform these operations, we create appropriate fields and methods in the **Xodim** class.

```
...
private Bo`lim bo`lim;
...
public void setBo`lim(Bo`lim d)
        {
            bo`lim = d;
        }
public Bo`lim getBo`lim()
        {
            return bo`lim;
        }
```

**Program code in C#:**

```
Bo`lim programmersBo`lim = new Bo`lim("Dasturchilar");
      programmersBo`lim.addXodim(Muhandis);
      Console.WriteLine("Bo'limdagi xodimlar: " +
Muhandis.getBo`lim().getNom());
```

### 2.3.1 ComLavozim

A comLavozim relation is a special case of aggregation. Through this relationship, organizational parts are considered as a component of a whole object. As a result of the loss of the whole object through the interaction between them, the components are also lost.

It is necessary to store information about the previous activities of the Xodim in the project. To do this, we create the **PastLavozim**class in the project. In this class, in addition to the **«name»** property, we create the **«Bo‘lim»** property for communication with the **Bo`lim**class.

Previous work activities of the Xodim in this organization are part of the information about the Xodim. Therefore, previous Lavozims cannot be created without using the **Xodim** class. The loss of the **Xodim** object should lead to the loss of the **PastLavozim** object (Fig. 7).

«PastLavozim» class:

```
public class PastLavozim{
       private String nomi;
       private Bo`lim bo`lim;
       public PastLavozim(String lavozim, Bo`lim dep){
          nomi = Lavozim;
          bo`lim = dep;
         }
       public void setNom(String newNom){
          nomi = newNom;
        }
       public String getNom(){
          return nomi;
         }
       public void setBo`lim(Bo`lim d){
                  bo`lim = d;
              }
      public Bo`lim getBo`lim(){
                  return bo`lim;
              }
```

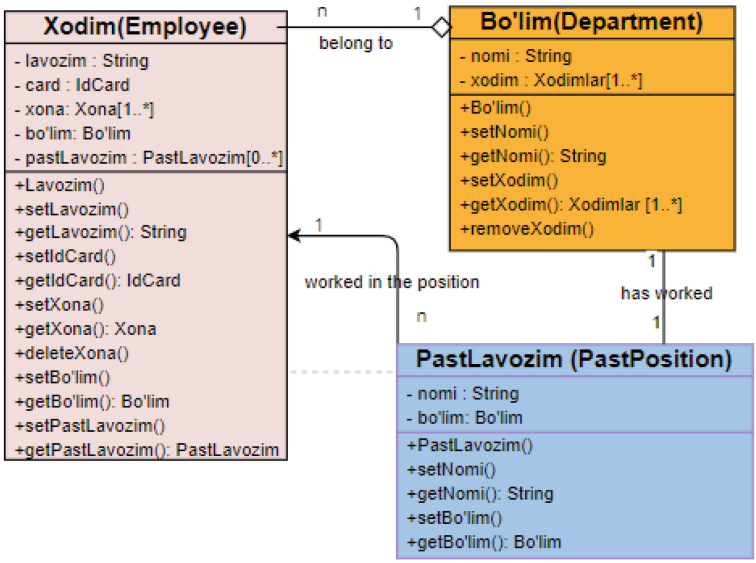


**Fig. 7.** ComLavozim relationship

We also create properties and methods corresponding to the **PastLavozim**object in the **Xodim** class:

```
...
List<PastLavozim> pastLavozim = new List<PastLavozim>();
...
public void setPastLavozim(PastLavozim p){
            pastLavozim.Add(p);
  }
  public List<PastLavozim> getPastLavozim(){
            return pastLavozim;
   }
  public void deletePastLavozim(PastLavozim p){
            pastLavozim.Remove(p);    }
...
```

**Program code in C#:**

```
// change of Lavozim
      Muhandis.setLavozim("Operator");
      // Let's look at the previous Lavozims:
      Console.WriteLine("Avvalgi lavozimlari:");
      foreach (PastLavozim pastLavozim in Muhandis.getPastLavozim()){
       Console.WriteLine(pastLavozim.getBo`lim());
       }
```

**Program code in C#:**

```
// change of Lavozim
      Muhandis.setLavozim("Operator");
      // Let's look at the previous Lavozims:
      Console.WriteLine("Avvalgi lavozimlari:");
      foreach (PastLavozim pastLavozim in Muhandis.getPastLavozim()){
       Console.WriteLine(pastLavozim.getBo`lim());
       }
```

3. Dependency

A dependency relationship generally represents a relationship (non-association, generalization, or implementation) between two elements (or two sets of elements) in a model. Changes made to one element, which are interrelated through a relationship of interdependence, must affect the second element.

The relationship between two elements is formed from one side to the other through an "arrow". In this relationship, the direction is from the client-class to the independent-class or source-class.

We use the **Menu**class to organize communication between the user and the information system (project). We create the **"showXodimlar"** method in the **Menu**class. In this method, a list of the organization's Xodimlar and their Lavozims is formed. An array of **Xodim** objects is passed as a method parameter. Therefore, changes made to the **Xodim**class may also require changes to the **Menu** class.

Noting that the **Menu**class does not belong to the practical field, it should be noted that it is one of the "system" classes of the software. Dependency relationship is shown in Fig. 8.

"Menu" class:

```
public class Menu
    {
        private static int i = 0;
        public static void showXodimlar(Xodim[] Xodimlar)
        {
            Console.WriteLine("Xodimlar ro'yxati:");
            for (i = 0; i < Xodimlar.Length; i++)
            {
                if (Xodimlar[i] is Xodim)
                Console.WriteLine(Xodimlar[i].getNom() + " - "
+ Xodimlar[i].getLavozim());
            } }}
```

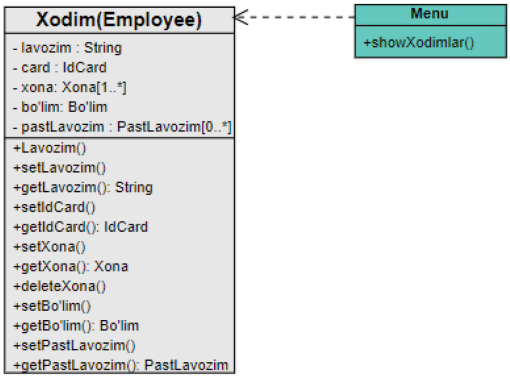


**Fig. 8.** Dependency relationship

**Program code in C#:**

**Program code in C#:**

```
   // add new Xodim
Xodim director = new Xodim("Akbarov", "Olim", "Direktor");
Menu menu = new Menu();
Xodim[] Xodimlar = new Xodim[10];
Xodimlar[0]= Muhandis;
Xodimlar[1] = director;
Menu.showXodimlar(Xodimlar);
```

## 4 Implementation

Implementation in C# is done as transparently as inheritance: an interface is declared and implemented in a class. We create the Unit interface to represent the implementation relationship. Given that the organization may have several Bo‘lims and branches, we will form an abstract unit with the **“Unit”** interface. Each unit has several Xodimlar. Therefore, it is important to determine the number of Xodimlar for each class that implements the **“Unit”** interface. Application attitude is shown in Fig. 9.

**Unit** interface:

```
public interface Unit
    {
        int getXodimCount();
    }
```

Application in "Bo`lim" class:

```
...
public int getXodimCount(){
                return getXodimlar().Count();
            }
```

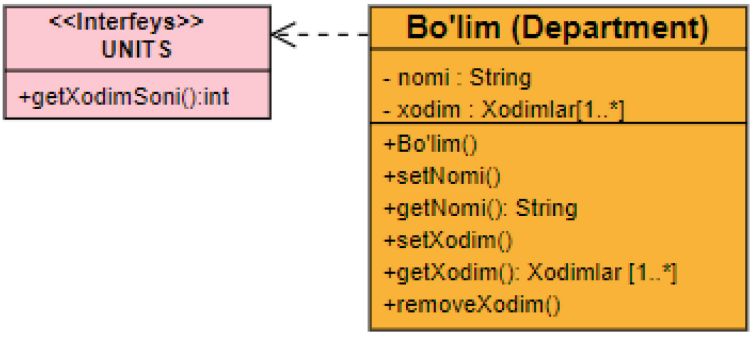


**Fig. 9.** Application attitude

**Program code in C#:**

```
Console.WriteLine(Muhandis.getBo`lim().getIsm() + " - bo'limda " +
Muhandis.getBo`lim().getPersonCount()+" ta xodim ishlaydi.");
```

It is not important for the **Bo`lim**class to implement the **«getPersonCount»** method. Because this class has a **«getXodimlar»** method that returns a list of **Xodim** objects.

Thus, as a result of modeling, the final diagram (Fig. 10) will look like this:

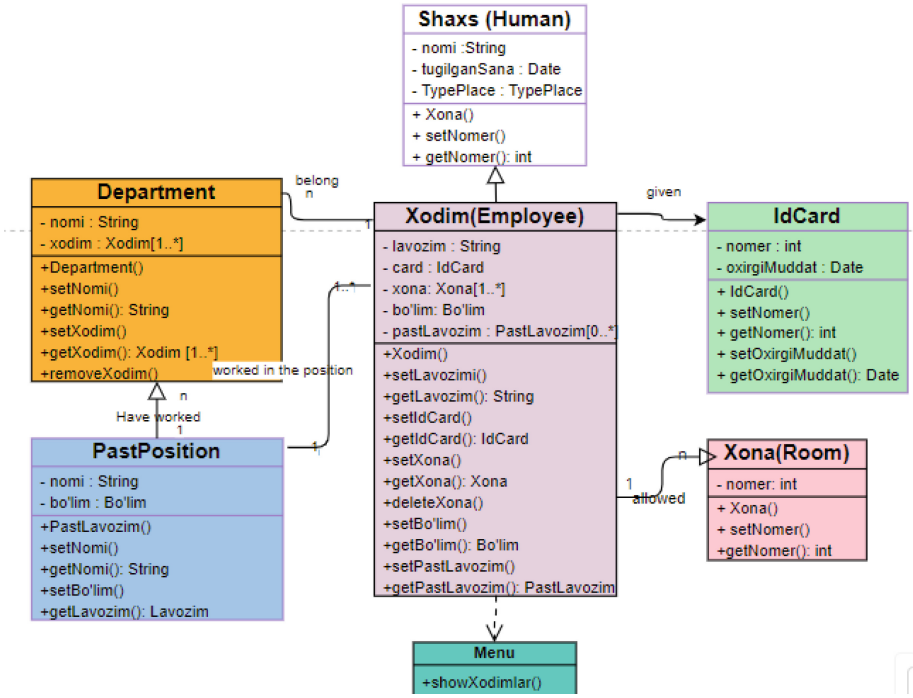


**Fig. 10.** Class diagram

## 4 Conclusion

The UML modeling language has tools for creating a class diagram, and in this article, the classes in the diagram are expressed in the C# object-oriented language. As a result of modeling, the final diagram was formed. Class diagrams provide a number of advantages for any organization. UML class diagrams can be used for the following purposes:

- modeling simple or complex business processes in information systems;
- better understanding of the overview of schemes in the information system;
- increasing the efficiency of visual representation and analysis of business processes of the information system;
- creating diagrams describing any special code required for information system development and implementation;
- description of connections between the components of the information system.

Based on the approach presented in this article, it is advisable to create class diagrams suitable for all objects and business processes of LMS.

## References


1. Chinosi, M., Trombetta, A.: BPMN: an introduction to the standard. Comput. Stand. Interfaces **34**(1), 124–134 (2012). https://doi.org/10.1016/j.csi.2011.06.002
2. Achraf, H.M., Redouane, E., el Mazoui Nadori Yasser, L.: Transforming the business process diagram into a class diagram by model-driven architecture. Indonesian J. Electr. Eng. Comput. Sci. **29**(2), 845 (2023). https://doi.org/10.11591/ijeecs.v29.i2.pp845-851
3. di Felice, P., Paolone, G., Paesani, R., Marinelli, M.: Design and implementation of a metadata repository about UML class diagrams. a software tool supporting the automatic feeding of the repository. Electronics (Switzerland) **11**(2), 201 (2022). https://doi.org/10.3390/electronics11020201
4. Berardi, D., Calvanese, D., de Giacomo, G.: Reasoning on UML class diagrams. Artif. Intell. **168**(1–2), 70–118 (2005). https://doi.org/10.1016/j.artint.2005.05.003
5. Bergström, G., et al.: Evaluating the layout quality of UML class diagrams using machine learning. J. Syst. Softw. **192**, 111413 (2022). https://doi.org/10.1016/j.jss.2022.111413
6. Decker, M.: An UML profile for the modelling of mobile business processes and workflows. In: Proceedings of the 5th International Conference on Mobile Multimedia Communications, MobiMedia 2009 (2009). https://doi.org/10.4108/ICST.MOBIMEDIA2009.8002
7. Egyed, A.: Consistent Adaptation and Evolution of Class Diagrams during Refinement. In: Wermelinger, M., Margaria-Steffen, T. (eds.) Fundamental Approaches to Software Engineering. FASE 2004. LNCS, vol. 2984. Springer, Berlin, Heidelberg (2004).https://doi.org/10.1007/978-3-540-24721-0_3
8. Massoni,T., Gheyi, R., Borba, P.: Formal Refactoring for UML Class Diagrams (2023).https://doi.org/10.5753/sbes.2005.23817
9. Haviluddin.: Memahami Penggunaan UML (Unified Modelling Language). Memahami Penggunaan UML (Unified Modelling Language) **6**(1) (2011)
10. Rumbaugh, J., Jacobson, I., Booch, G.: El Lenguaje Unificado de Modelado. Elements (1999)
11. Martin, R.C.: UML Tutorial: Part 1 -- Class Diagrams. Analysis 108 (1997)
12. Rumbaugh, J., Jacobson, I., Booch, G.: The UML reference manual. In: New York: Addison-Wesley (Vol. 1) (1999)
13. Object Management Group. OMG Unified Modeling Language TM (OMG UML). InformatikSpektrum, 21(May) (2011)
14. Ma, Z., He, H., Liu, J., He, X.: Analysis of the evolution of the UML metamodel. In: MODELSWARD 2018 - Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, 2018-January (2018). https://doi.org/10.5220/0006571303560363
15. Elov B.B., Primova, M.: Using BPMN methodology in business processes of SMART LMS. Acta NUUz. - Tashkent, 2017, - № 2/2
16. Sirojiddinov Sh. S., Elov, B.B.: NATIONAL UNIVERSITY OF UZBEKISTAN ON THE THRESHOLD TO SMART UNIVERSITY. International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR) ISSN(P): 2249–6831; ISSN(E): 2249–7943 Vol. 6, Issue 2, Apr 2016, 23–28 © TJPRC Pvt. Ltd.

# Author Index




S. Manoharan et al. (Eds.): ICAIS 2024, ISEM 4, pp. 511–513, 2024.
https://doi.org/10.1007/978-3-031-61475-0