МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**О Т Ч Ё Т**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

**«ООП в C#. Повторное использование кода. Наследование.»**

**по курсу:**

***"Конструирование программного обеспечения"***

|  |  |
| --- | --- |
| Выполнили, студенты гр. 10701121: | Качан И. В.  Длусский К. Ю. |
|  |  |
| Проверил: | Станкевич С. Н. |

Минск – 2023

**Цель работы:**

Углубить свои фундаментальные знания в использовании методологии ООП, а также научиться практически применять инструменты языка C# для повторного использования кода в виде ассоциации, наследования, агрегации, композиции и делегирования.

**Требования**:

1) Необходимо скорректировать UML-диаграмму взаимодействия классов и объектов программной системы с учётом вносимых дополнений и изменений.

2) При корректировки программной системы необходимо полностью использовать своё ООП-воображение и по максимум использовать возможности, которые предоставляет язык Java для программирования с использованием методологии ООП.

3) Каждый пользовательский тип должен иметь адекватное осмысленное имя и информативный состав (соответствующие конструкторы: по умолчанию, с параметрами, конструктор-копирования; get- и set-методы для доступа к состоянию объекта; корректно переопределённые методы базового класса Object: toString(), equals(), hashCode() и др.).

4) Также рекомендуется придерживаться Single Responsibility Principle, SRP (принципа единственной ответственности): у каждого пакета, класса или метода должна быть только одна ответственность (цель), т.е. должна быть только одна причина изменить в дальнейшем соответствующий блок кода.

5) Добавляемые классы необходимо грамотно разложить по соответствующим пакетам, которые должны иметь «адекватные» названия и быть вложены в указанные стартовые пакеты: by.bntu.fitr.poisit.nameofstudent.nameofproject. Basic of OOP in Java. Code Reuse. Inheritance 2 | We hope you enjoy working with Java!

6) В соответствующих важных компонентах программной системы необходимо предусмотреть «защиту от дурака».

7) В качестве хранилище данных использовать Java-массивы!!! Все контейнерный класс должны поддерживать расширяемость, т.е. они не ограничены в объёме хранимых данных и динамически должны расширяться.

8) Весь код модели системы должен быть покрыт модульными тестами, подтверждающие качество и работоспособность ядра системы. Необходимо проверить все тестовые случаи. Для реализации модульных тестов рекомендуется использовать jUnit фреймворк.

9) При разработке программ придерживайтесь соглашений по написанию кода на Java (Java Code-Convention) !!!

**Основное задание**:

Необходимо решить задачу с использованием методологии ООП. Для чего  
необходимо подобрать самостоятельно соответствующую проблемную  
(предметную/доменную) область, которая базируется на объектах и событиях  
реального мира (примеры соответствующих предметных областей приведены  
ниже). Спроектировать классы (собственные пользовательские типы данных)  
в языке Java для программного представления данных объектов и основной  
логики будущей программной системы.  
Система должна решать, как минимум, два полезных действия и иметь дополнительно следующие опции:  
− не менее 3 разнообразных классов предметной области;  
− не менее 5 атрибутов (состояния) и методов (поведения) в классе-сущности;  
− не менее 3 методов, которые реализуют бизнес-логику программы, в соответствующих функциональных классах;  
− хранить глобальные характеристики системы или характеристики уровня  
отдельных классов.  
На базе спроектированной программной системы реализовать программу и  
продемонстрировать её работоспособность.

**Предметная область:**

Компания по разработки программного обеспечения, IT-компания  
(Software Company). Есть IT-компании, которые состоят из соответствующих  
сотрудников. Каждый сотрудник за свою работу получает соответствующую  
зарплату. Необходимо найти общий фонд заработной платы компании на месяц, а также определить сотрудников с максимальной (минимальной) заработной платой

**Результаты выполнения программы:**

**Изображение выглядит как текст

Автоматически созданное описание**

**Рисунок 1. Добавление сотрудников**

**Изображение выглядит как текст

Автоматически созданное описание**

**Рисунок 2. Вывод всех сотрудников и их суммарная зарплата за месяц.**

**Изображение выглядит как текст

Автоматически созданное описание**

**Рисунок 3. Вывод сотрудников с минимальной и максимальной зарплатой.**

**Изображение выглядит как текст

Автоматически созданное описание**

**Рисунок 4. Повышение зарплаты сотруднику.**

**UML-диаграмма классов:**

Изображение выглядит как диаграмма

Автоматически созданное описание

Изображение выглядит как диаграмма

Автоматически созданное описание

**Рисунок 5. UML-диаграмма классов.**

**Листинг исходного кода программы:**

**Класс Employee.cs**

using System;

using System.Collections.Generic;

namespace KPOlaba1

{

abstract class Employee

{

private string name;

private double salary;

private string rank;

private double salaryRatio;

protected static double startSalary = 500;

public static Dictionary<String, double> rankDict = new Dictionary<string, double>{ { "junior", 0 }, { "middle", 0 }, { "senior", 0 } };

public string Name

{

get => name;

set => name = value;

}

public virtual double Salary

{

get => salary;

set => salary = value;

}

public virtual double SalaryRatio

{

get => salaryRatio;

set => salaryRatio = value;

}

public string Rank

{

get => rank;

set => rank = value;

}

public static double StartSalary

{

get => startSalary;

set => startSalary = value;

}

public Employee(string name, string rank)

{

Name = name;

Rank = rank;

}

public Employee() { }

public abstract void PrintInfo(); // вывод информации о сотруднике

}

}

**Класс SoftwareEngineer.cs**

using System;

using System.Collections.Generic;

namespace KPOlaba1

{

internal class SoftwareEngineer : Employee

{

public SoftwareEngineer(string name, string rank) : base(name, rank)

{

rankDict = new Dictionary<String, double>() { { "junior", 2 }, { "middle", 2.5 }, { "senior", 4 } };

SalaryRatio = rankDict[Rank];

Salary = startSalary \* SalaryRatio;

}

public SoftwareEngineer() : base() { }

public override void PrintInfo() // вывод информации о сотруднике

{

Console.WriteLine("--------------------");

Console.WriteLine("Engineer info:");

Console.WriteLine("Name: {0} Rank: {1} Salary {2}", Name, Rank, Salary);

Console.WriteLine("--------------------\n");

}

}

}

**Класс QAengineer.cs**

using System;

using System.Collections.Generic;

namespace KPOlaba1

{

internal class QAengineer : Employee

{

public QAengineer(string name, string rank) : base(name,rank)

{

rankDict = new Dictionary<String, double>() { { "junior", 1 }, { "middle", 1.5 }, { "senior", 2 } };

SalaryRatio = rankDict[Rank];

Salary = startSalary \* SalaryRatio;

}

public QAengineer() : base() { }

public override void PrintInfo() // вывод информации о сотруднике

{

Console.WriteLine("--------------------");

Console.WriteLine("QA Engineer info:");

Console.WriteLine("Name: {0} Rank: {1} Salary {2}", Name, Rank, Salary);

Console.WriteLine("--------------------\n");

}

}

}

**Класс ListOfEmployeers.cs**

using System;

using System.Collections.Generic;

using System.Web.UI;

using System.Xml.Linq;

namespace KPOlaba1

{

abstract class ListOfEmployeers

{

public static List<Employee> empoyees = new List<Employee>();

public static void AddEmployee() // добавить сотрудника

{

Console.Write("Name: ");

string name = Console.ReadLine();

Console.Write("Rank(junior, middle, senior): ");

string rank = Console.ReadLine();

if (Employee.rankDict.ContainsKey(rank) == false)

{

Console.WriteLine("Invalid input");

return;

}

Console.Write("Speciality(QA, Engineer): ");

string speciality = Console.ReadLine();

switch (speciality)

{

case "Engineer":

empoyees.Add(new SoftwareEngineer(name, rank));

break;

case "QA":

empoyees.Add(new QAengineer(name, rank));

break;

default:

Console.WriteLine("Invalid input\n");

break;

}

}

public static void DeleteEmployee() // удалить сотрудника

{

Console.WriteLine("Enter a name: ");

string name = Console.ReadLine();

if (empoyees.Count != 0)

{

foreach (Employee empoyee in empoyees)

{

if (empoyee.Name == name)

{

empoyees.RemoveAll(x => x.Name == name);

Console.WriteLine("Success\n");

break;

}

else

{

Console.WriteLine("There is no employee with this name\n");

}

}

}

else

{

Console.WriteLine("List of emloyeers is empty\n");

}

}

public static Pair FindMaxMinSalary() // поиск максимальной и минимальной зарплаты

{

List<int> min = new List<int>();

List<int> max = new List<int>();

Pair pair = new Pair();

int indexMin = 0, indexMax = 0;

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

{

if (empoyees[indexMax].Salary < empoyees[i].Salary)

{

max.Clear();

indexMax = i;

max.Add(indexMax);

continue;

}

else

{

if (empoyees[indexMax].Salary == empoyees[i].Salary)

{

max.Add(i);

continue;

}

}

}

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

{

if (empoyees[indexMin].Salary > empoyees[i].Salary)

{

min.Clear();

indexMin = i;

min.Add(indexMin);

continue;

}

else

{

if (empoyees[indexMin].Salary == empoyees[i].Salary)

{

min.Add(i);

continue;

}

}

}

pair.First = min;

pair.Second = max;

return pair;

}

public static void PrintMinMax() // вывести людей с максимальной и минимальной зарплатой

{

Console.WriteLine("Info about employeers with max and min salary:");

Pair pair = FindMaxMinSalary();

List<int> min = (List<int>)pair.First;

List<int> max = (List<int>)pair.Second;

Console.WriteLine("Minimum salary: ");

foreach (int indexMin in min)

{

empoyees[indexMin].PrintInfo();

}

Console.WriteLine("---------");

Console.WriteLine("Maximum salary: ");

foreach (int indexMax in max)

{

empoyees[indexMax].PrintInfo();

}

Console.WriteLine("---------");

}

public static void RaiseSalary() // увеличить/уменьшить зарплату

{

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

string name = Console.ReadLine();

Console.Write("Enter amount of money: ");

double.TryParse(Console.ReadLine(), out double amount);

foreach (Employee empoyee in empoyees)

{

if (empoyee.Name == name)

{

empoyee.Salary += amount;

}

}

}

public static void PrintEmployeersInfo() // вывести всех сотрудников

{

Console.WriteLine("List of employeers:\n");

double employeeMonthSalary = 0;

foreach (Employee empoyee in empoyees)

{

employeeMonthSalary += empoyee.Salary;

empoyee.PrintInfo();

}

Console.WriteLine("\n\nEmployees salary for a month: {0}", employeeMonthSalary);

}

public static void ChangeStartSalary() // изменить стартовую зарплату

{

bool isCorrect = true;

Console.WriteLine("Enter new start salary: ");

isCorrect = double.TryParse(Console.ReadLine(), out double newStartSalary);

if (isCorrect == true)

{

foreach (Employee employee in empoyees)

{

employee.Salary = employee.Salary \* (newStartSalary / Employee.StartSalary);

}

Employee.StartSalary = newStartSalary;

}

else

{

Console.WriteLine("Invalid Input");

}

}

}

}

**Класс Program.cs**

using System;

using System.Collections.Generic;

using System.Web.UI;

/\*

\* Console application "IT Company"

\* CEO - Kachan Ivan, CTO - Dlussky Kirill. Group 10701121

\* 23.03.2023

\* version 1.0

\*/

namespace KPOlaba1

{

internal class Program

{

static void Main(string[] args)

{

bool isExit = false;

while (isExit == false)

{

Console.WriteLine("Select an action:");

Console.WriteLine("1. Add new Employee\n2. Delete employee\n3. Raise employee salary\n4. Print list of employees\n5. Find employee with MAX and MIN salary" +

"\n6. Change start salary\n7. Exit");

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

switch (input)

{

case 1:

ListOfEmployeers.AddEmployee();

break;

case 2:

ListOfEmployeers.DeleteEmployee();

break;

case 3:

ListOfEmployeers.RaiseSalary();

break;

case 4:

ListOfEmployeers.PrintEmployeersInfo();

break;

case 5:

ListOfEmployeers.PrintMinMax();

break;

case 6:

ListOfEmployeers.ChangeStartSalary();

break;

case 7:

isExit = true;

break;

default:

Console.WriteLine("\nInvalid input! Try again\n");

break;

}

Console.WriteLine("\n\n");

}

}

}

}

**Контрольные вопросы**

1. Что такое повторное использование кода?

2. Какие подходы и средства предлагает методология ООП для повторного использования кода?

3. На что указывает принцип DRY (Don't Repeat Yourself)?

4. Что такое зависимость? Как зависимость реализуется в языке C#?

5. Что такое ассоциация? Как реализована ассоциация в языке C#?

6. Что такое наследование? Как реализовано наследование в языке C#?

7. Какие преимущества и недостатки есть у множественного наследования

классов?

8. Разрешено ли в языке C# множественное наследование? Если разрешено

множественное наследование в языке C#, то как наследуются дочерним

классом одинаковые характеристики из разных базовых классов?

9. Как при наследовании вызываются конструкторы?

10. Зачем нужно в языке C# ключевое слово super?

11. Кто находится во главе всей иерархии в языке C#?

12. Какой именно минимальный функционал получают все экземпляры классов в

языке C#?

13. Опишите функционал базового класса Object для всего ООП-мира в языке

C#.

14. Что такое агрегация? Как реализована агрегация в языке C#?

15. Что такое композиция? Как реализована композиция в языке C#?

16. Что такое делегирование? Как реализовано делегирование в языке C#?

17. Основное отличие наследования от агрегации/композиции?

18. Преимущества и недостатки композиции и агрегации? Преимущества и

недостатки наследования?

19. Как с помощью языка UML отобразить все зависимости между классами на

UML-диаграмме классов?

20. Какой должен базовый класс у классов-исключений? Как создавать

пользовательские исключения?

**Ответы на контрольные вопросы**

1)Повторное использование кода (code reuse) - это процесс использования уже написанного кода в новых проектах или функциях для уменьшения объема работы по написанию нового кода и повышения эффективности разработки.

2)Методология ООП предлагает несколько подходов и средств для повторного использования кода, таких как наследование, композиция, агрегация, делегирование и шаблоны проектирования.

3)Принцип DRY (Don't Repeat Yourself) указывает на необходимость избегания дублирования кода и сведения к минимуму повторения одинаковых участков кода в разных местах программы. Это позволяет уменьшить вероятность ошибок и упрощает поддержку кода.

4)Зависимость (dependency) - это связь между двумя объектами, когда изменения в одном объекте влияют на другой объект. В языке C# зависимость может быть реализована с помощью использования одного объекта в качестве параметра, аргумента или свойства другого объекта.

5)Ассоциация (association) - это связь между двумя объектами, когда один объект использует другой объект, но не является его частью. В языке C# ассоциация может быть реализована с помощью создания объектов других классов внутри класса.

6)Наследование (inheritance) - это механизм ООП, позволяющий создавать новый класс на основе уже существующего класса, заимствуя его свойства и методы. В языке C# наследование реализуется с помощью ключевого слова "class" и наследования от базового класса.

7)Преимущества множественного наследования - возможность использования кода из нескольких базовых классов. Недостатки множественного наследования - увеличение сложности кода и возможность конфликтов имен при наследовании от нескольких классов.

8)В языке C# множественное наследование запрещено. Вместо этого используется множественное наследование интерфейсов.

9)При наследовании конструктор базового класса вызывается перед конструктором дочернего класса.

10)Ключевое слово "super" используется в других языках программирования для вызова методов родительского класса. В языке C# используется ключевое слово "base".

11)Все классы в C# наследуются от базового класса Object, который находится во главе всей иерархии.

12)Все экземпляры классов в C# получают минимальный функционал, определенный в базовом классе Object, включая методы, такие как ToString(), GetHashCode(), Equals() и GetType().

13)Базовый класс Object в C# предоставляет функционал, который доступен всем объектам в языке. Это включает в себя методы, такие как ToString(), GetHashCode(), Equals() и GetType(), а также виртуальные методы Finalize() и MemberwiseClone(). Кроме того, этот класс определяет операторы равенства и неравенства и операторы приведения типов.

14)Агрегация - это отношение между двумя классами, где один класс содержит ссылку на другой класс в качестве члена. В C# это обычно реализуется через композицию объектов. Например, если класс A содержит ссылку на объект класса B, то это может быть реализовано как поле типа B в классе A.

15)Композиция - это отношение между двумя классами, где один класс содержит другой класс как часть своей реализации. В C# это обычно реализуется через создание объектов других классов внутри текущего класса. Например, если класс A содержит объект класса B, то это может быть реализовано как поле типа B в классе A и создание объекта типа B в конструкторе класса A.

16)Делегирование - это механизм, позволяющий передавать вызов метода из одного объекта другому объекту, который занимается его выполнением. В C# делегирование реализуется через использование делегатов.

17)Основное отличие наследования от агрегации/композиции заключается в том, что при наследовании класс-наследник получает доступ к методам и полям базового класса, тогда как при агрегации/композиции объект класса содержит ссылку на другой объект класса или создает его внутри себя.

18)Преимущества и недостатки композиции и агрегации:

Композиция:

• Преимущества:

o Более жесткая связь между объектами, что обеспечивает более явное и понятное поведение;

o Возможность создавать сложные объекты, состоящие из других объектов;

o Возможность изменять состав объектов во время выполнения программы;

o Улучшенная изоляция и защита объектов.

• Недостатки:

o Ограниченность гибкости, которую предоставляет наследование;

o Более сложная реализация, чем у наследования.

Агрегация:

• Преимущества:

o Более слабая связь между объектами, что позволяет менять объекты без серьезных последствий;

o Упрощенная реализация.

• Недостатки:

o Увеличенная сложность для понимания взаимодействия между объектами;

o Ограниченность в возможности изменять состав объектов.

19)Для отображения зависимостей между классами на UML-диаграмме классов используется стрелка со связью, указывающей на зависимый класс. Это означает, что класс, на который указывает стрелка, зависит от класса, из которого исходит стрелка.

20)В языке C# базовым классом для классов-исключений является класс Exception. Для создания пользовательского исключения необходимо создать класс, производный от класса Exception, и реализовать необходимые конструкторы и свойства для хранения информации об исключении. Класс пользовательского исключения должен иметь имя, отражающее тип исключения, и должен быть доступен в области видимости, в которой будет вызвано исключение.

**Вывод:**