МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чернігівський національний технологічний університет

СТВОРЕННЯ ДЕРЕВОВИДНИХ СТРУКТУР ДАНИХ ЗАСОБАМИ JAVA

МЕТОДИЧНІ ВКАЗІВКИ
до виконання розрахунково-графічної роботи з дисципліни
« Об'єктно-орієнтоване програмування»
для студентів спеціальності
128 – "Комп'ютерна інженерія"

ЗАТВЕРДЖЕНО
на засіданні кафедри
інформаційних та комп'ютерних систем
протокол № 1 від 29.08.18

Чернігів ЧНТУ 2018

Створення деревовидних структур даних засобами Java. Методичні вказівки до виконання розрахунково-графічної роботи з дисципліни «Об’єктно-орієнтоване програмування» для студентів спеціальності 128 – „Комп’ютерна інженерія”. /Укл.: Бивойно П.Г., Бивойно Т.П. – Чернігів: ЧНТУ, 2018. – 29 с.

Укладачі: Бивойно Павло Георгійович, доцент, к.т.н.,
Бивойно Тарас Павлович, старший викладач

Відповідальний за випуск: С.В. Зайцев, доктор. техн. наук, зав. кафедри інформаційних комп'ютерних систем Чернігівського національного технологічного університету

Рецензент: В.А. Бичко, канд. ф-м. наук, доцент кафедри інформаційних і комп'ютерних систем Чернігівського державного технологічного університету

ЗМІСТ

Вступ

Поняття розрахунково-графічна робота (РГР) прийшло у вищу школу з дисциплін навчальних планів інженерно-механічних спеціальностей. У межах розрахунково-графічної роботи студент повинен був виконати деякі розрахунки і графічні побудови (креслення, ескіз, діаграму), що були необхідні для вирішення деякої інженерної проблеми. До того ж часто використовувалися так звані графоаналітичні методи розрахунків. Тому назва «розрахунково-графічна робота» була, безсумнівно, доречною.

Проте потім, завдання студентам, що були пов'язані з вирішенням проблем інших галузей знань, теж почали називати розрахунково-графічними роботами. З'явилися РГР з хімії, бухгалтерського обліку та інші. У цих завданнях, зазвичай, ніяких графічних елементів вже не було, залишалися тільки розрахунки, а інколи і їх не було, та назва зберіглася.

У курсі «Об'єктно-орієнтоване програмування» виконання розрахунково-графічної є частиною самостійної роботи студентів над дисципліною. РГР передбачає проведення об'єктно-орієнтованого аналізу деякої предметної області і подальшу побудову програми, що вирішує деякі проблеми цієї області. Ніяких розрахунків робота не передбачає, але має буди програмна реалізація додатку.

Окрім аналізу предметної області студент має проаналізувати можливості Java та визначити перелік інтерфейсів та класів, що можуть бути використані або безпосередньо, або як батьківські класи для реалізації класів майбутнього додатку. У процесі програмної реалізації мають бути візуальна композиція та класи, що відповідають складовим частинам предметної області

Результатом розрахунково-графічної роботи є звіт обсягом приблизно 15 сторінок друкованого тексту оформленого відповідно до стандартів кафедри. Бали за розрахунково-графічну роботу виставляються з урахуванням своєчасності та якості виконання і включаються як складова до результатів семестру.

За звичай номер варіанту завдання для РГР призначається викладачем, та студент може і сам обрати предметну область для реалізації, але завдання обов'язково має бути погоджено з викладачем

1 ЗАВДАННЯ ДО РОЗРАХУНКОВО - ГРАФІЧНОЇ РОБОТИ

Варіант	Можливі рівні		
1. Мінтранспорту	місто	вокзал	рейс
2. Міські автовокзали	вокзал	напрям	рейс
3. Аеросвіт	місто	аеропорт	рейс
4. Торгівля комп'ютерами	країна	фірма	комп'ютер
5. Торгівля оргтехнікою	місто	магазин	товар
6. Торгівельна фірма	склад	група товарів	товар
7. Країна	регіон	область	місто
8. Управління екології	область	район	екопроблеми
9. Бюро інвентаризації	вулиця	будинок	квартира
10. Міськкомунхоз	жек	професія	робітник
11. Олімпіада	країна	вид спорту	спортсмен
12. Федерація футболу	клуб	амплуа	спортсмен
13. Чемпіонати	коли і де	види	учасники
14. Хозчастина	корпус	аудиторія	мебель
15. Фармацевтична фірма	країна	відділ	ліки
16.Торгова мережа	фірма	постачальники	товари
17. Підприємство	вироби	вузли	деталі
18. Деканат1	кафедра	предмет	література
19. Деканат2	спеціальність	група	студент
20. Деканат3	кафедри	викладачі	предмети
21. Ресторан	кухня	блюдо	продукті
22. Ремонтна майстерня	виріб	вид ремонту	деталі
23. Космічне бюро	зірки	планети	супутники
24. Товарна база	товари	клієнти	накладна
25. ООН	континенти	регіони	держави
26. Бюро захисту природи	Кліматичні зони	Рослини, тварини	представники
27. Мережа магазинів	магазин	персонал	товари
28. Завод	цех	обладнання	робітники

2 ВИМОГИ ДО РОБОТИ

2.1 Загальні вимоги

РГР передбачає створення динамічної інформаційної структури даних у вигляді багаторівневого розгалуженого дерева, на основі стандартної моделі для дерева з бібліотеки Java, та візуального додатку, що забезпечить роботу із структурою даних.

Кількість рівнів дерева має бути не менше чотирьох разом з коренем. Компоненти кожного рівня мають бути об’єктами, що містять не менше трьох полів. Допускається зменшувати кількість полів на одному рівні за рахунок збільшення кількості полів на інших рівнях. Поля об’єктів не можуть бути одного типу. Принаймні на двох рівнях у об’єктів мають бути числові поля.

Додаток має забезпечити реалізацію таких функцій:

- відобразити перелік усіх компонентів у вигляді дерева;
- додати компонент до вибраного вузла дерева;
- вилучити компонент із дерева;
- змінити дані вибраного компоненту;
- відшукати компонент за якоюсь ознакою;
- обчислити якусь числову характеристику групи елементів;
- зберегти структуру даних у файлі;
- відновити структуру даних із файлу.

Предметна область роботи має відповідати одному з варіантів із наведеного нижче переліку. Номер варіанту студент отримує у викладача. Студент може запропонувати і свій варіант.

Детальні вимоги до роботи наводяться у технічному завданні, яке складає студент і узгоджує з викладачем.

На захист студент має представити працюючий додаток і пояснювальну записку, оформлену відповідно до стандартів кафедри.

2.2 Обов’язкові розділи пояснювальної записки

Пояснювальна записка до роботи створюється відповідно до вимог кафедри викладених у методичних вказівках [1] (СОККР-2002). Наведені нижче пункти можна вважати доповненням до вимог СОККР-2002.

2.2.1 Технічне завдання

У технічному завданні має бути наведено опис системи, для якої створюється модель. Зокрема, мають бути перелічені поля об’єктів на усіх рівнях. Також мають бути сформульовані завдання для пошуку у дереві та завдання для визначення числових характеристик.

2.2.2 Аналіз та розробка системи що підлягає реалізації

У цьому розділі слід навести схематичне зображення предметної області у вигляді діаграми класів, а також короткі пояснення до неї.

2.2.3 Реалізація системи

У цьому розділі слід навести тексти класів, що було створено. Обов’язково мають бути коментарі до методів.

2.2.4 Тестування системи

У цьому розділі слід навести копії екранів або консолі з результатами роботи системи у різних режимах роботи.

3 ТЕОРЕТИЧНІ ВІДОМОСТІ

3.1 Класи Java для роботи з деревом

Стандартна бібліотека Java надає усі необхідні інструменти для створення та обробки деревовидних структур даних. Відповідні класи та інтерфейси можна знайти у пакетах javax.swing та javax.swing.tree.

3.1.1 Клас javax.swing.JTree

Об’єкти класу JTree призначені для відображення ієрархічних даних. Слід розуміти, що об'єкт JTree насправді не містить дані користувача, він просто забезпечує відображення цих даних. Як і багато інших компонентів Swing, цей компонент отримує дані при зверненні до своєї моделі даних. Приклад вигляду дерева наведено на рисунку 3.1.

Рисунок 3.1 – Вигляд об’єкту класу JTree

Як видно з рисунку, JTree показує свої дані по вертикалі. Кожен рядок дерева, що зображено, містить тільки один елемент даних, який називається вузлом. Кожне дерево має кореневий вузол, від якого все вузли спускаються вниз. Вузол може або мати дітей або ні. Вузли, які не мають дітей є листовими вузлами. Дітей вузлів з розгалуженнями можна робити або невидимими, як на рисунку 3.1, або видимими.

Конкретний вузол в дереві може бути ідентифікований або за допомогою об'єкту TreePath, який інкапсулює вузол і всіх його предків, або його рядком у відображенні.

Для програміста, що створює інтерфейс користувача, найбільшу цікавість представляє метод getLastSelectedPathComponent(), що повертає посилання на вузол дерева, що вибрав користувач. Метод повертає посилання типу Object, а реально вузли можуть мати типи або визначені в Java, або це можуть бути типи, що визначені розробником моделі дерева.

3.1.2 Інтерфейс javax.swing.tree.TreeModel

Компонент JTree може відображати будь яку модель, якщо вона реалізує інтерфейс TreeModel. Інтерфейс оперує з поняттями кореня дерева, вузла, дітей вузла. Методи інтерфейсу мають забезпечити доступ до цих складових.

Перелік цих методів можна знайти у файлі з вихідним текстом класу.

Якщо компонент JTree створюється з простим конструктором, то в якості моделі використовується модель за замовчуванням, яка створюється у самому класі. Саме ця модель відображена на рисунку 3.1.

Якщо потрібно передати компоненту іншу модель, слід використовувати конструктор з параметром типу TreeModel.

Для створення моделі можна скористатися конструктором класу DefaultTreeModel куди в якості параметра передати посилання на корінь дерева. Корінь дерева у цьому випадку має бути типу javax.swing.tree.TreeNode.

Окрім того, можна проблему побудови моделі передати об’єкту класу JTree, скориставшись його конструктором з параметром типу TreeNode, який сприймається цим конструктором як корінь дерева.

Отже, проблема створення моделі дереві фактично зводиться до створення класу вузла дерева, що реалізує інтерфейс TreeNode.

3.1.3 Інтерфейс javax.swing.tree.TreeNode

Цей інтерфейс визначає вимоги до об’єкту, що може бути вузлом дерева, що відображується за допомогою компонента JTree. Перелік методів цього інтерфейсу наведено на рисунку 3.3.

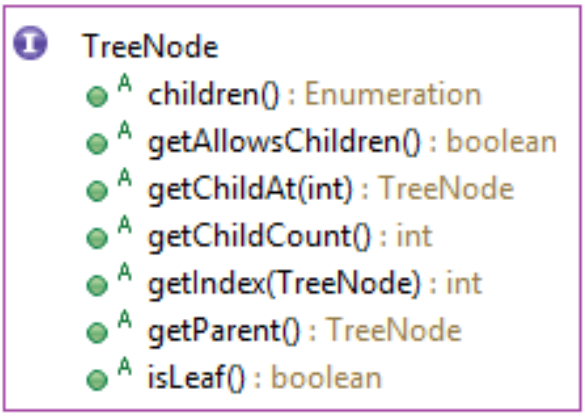

Рисунок 3.3 – Перелік методів інтерфейсу TreeNode

Більшість методів інтерфейсу відповідає вимогам інтерфейсу TreeModel і мають такі самі назви. Ці методи та метод getParent() зрозумілі без пояснень.

Метод getAllowsChildren() має повертати true, якщо вузлу дозволено мати нащадків.

Метод children() має повертати ітератор типу Enumeration, який надає послідовний доступ до синів вузла.

Реалізація методів цього інтерфейсу дає можливість відобразити дерево у компоненті JTree, але інтерфейс не визначає методи за допомогою яких можна додавати дітей до вузла, вилучати їх і таке інше. Ці методі мають бути визначені та реалізовані у класах користувача дерева, або для створення класу використовувати розширення інтерфейсу TreeNode.

3.1.4 Інтерфейс javax.swing.tree.MutableTreeNode

Інтерфейс MutableTreeNode, рисунок 3.3, є розширенням інтерфейсу TreeNode. Методи цього інтерфейсу передбачають можливість працювати з вузлом і з синами вузла використовуючи індекси та посилання.

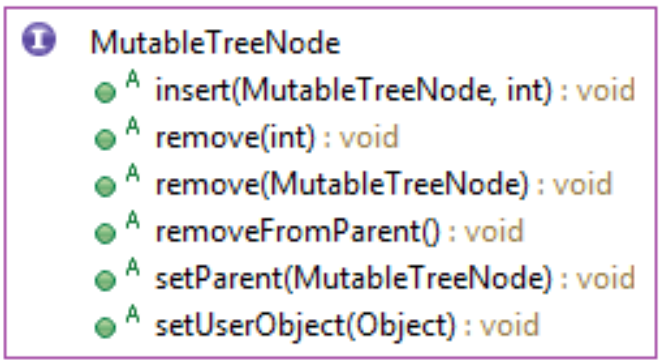

Рисунок 3.3 – Перелік методів інтерфейсу MutableTreeNode

3.1.5 Клас javax.swing.tree.DefaultMutableTreeNode

Цей клас реалізує інтерфейси TreeNode та MutableTreeNode. Окрім методів цих інтерфейсів клас пропонує чималу низку додаткових методів, які спрощують роботу з деревом. Наведемо тут тільки деякі з них.

Метод add(MutableTreeNode) додає нового сина у кінець списку.

Методи getFirstChild(), getLastChild(), getChildAfter(TreeNode), getChildBefore(TreeNode), removeAllChildren() надають додаткові можливості роботи з переліком синів.

Методи getSiblingCount(), isNodeSibling(TreeNode), getNextSibling(), getPreviousSibling() дозволять роботу з братами вузла.

Методи getFirstLeaf(), getLastLeaf(), getLeafCount(), getNextLeaf(), getPreviousLeaf() надають можливість обробляти листи дерева.

Методи preorderEnumeration() та postorderEnumeration() повертають ітератори типу Enumeration для проходу по дереву у прямому та зворотному напрямах.

Метод pathFromAncestorEnumeration(TreeNode) повертає ітератор типу Enumeration для проходу від заданого предку до вузла this.

Методи isNodeDescendant(DefaultMutableTreeNode), isNodeChild(TreeNode), isNodeAncestor(TreeNode), isNodeRelated(DefaultMutableTreeNode) дозволяють з’ясовувати родинні зв’язки між вузлами.

Таким чином для створення моделі дерева можна скористатися класом DefaultMutableTreeNode. Щоб створити корінь та вузли дерева можна використовувати конструктор цього класу з параметром типу Object, через який передавати вузлу дані, що мають зберігатися у створюваному вузлу. Додавати дітей до вузла можна за допомогою методу add, або через метод insert, а вилучати за допомогою одного з методів remove. Для отримання переліку синів можна скористатися ітератором children, а для обходу усього дерева – ітераторами preorderEnumeration()або postorderEnumeration().

4 ПРИКЛАД ПОБУДОВИ СИСТЕМИ

4.1 Технічне завдання

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання РГР з дисципліни "ООП"
студента Камко П.С., група ПІ-341

Тема роботи: Інформаційна система "Університет".

Очікувані технічні та експлуатаційні результати роботи:

Java проект, який забезпечує роботу з інформацією про міністерство освіти (університети, факультети, кафедри) згідно варіанта завдання 0.

Корінь «університет» має містити назву університету, ім’я ректора, рівень акредитації.

Рівень 1 - «факультет», зберігає інформацію про назву факультету, ім’я декана та номер корпусу, де розташований факультет. Факультети впорядковуються відповідно до назви.

Рівень 2 - «кафедра», містить назву кафедри, ім’я завідувача, та ознаку - чи є кафедра випускаючою.

Рівень 3 «викладач», містить ім’я викладача, посаду, заробітну плату.

Окрім типових операцій введення даних, їх корегування, вилучення та перегляду, мають бути реалізовані такі специфічні операції:

- пошук викладача, найстаршого за віком;
- підрахунок та виведення на консоль загальної кількості випускаючих кафедр для кожного факультету.

Планова трудомісткість роботи: 0.5 кредите **(**15 годин**).**

Обсяг текстової документації:

Пояснювальна записка до проекту об'ємом 15-20 сторінок друкованого тексту формату А4 і програмна документація на систему обсягом 35-40 сторінок друкованого тексту формату А4. Обсяги текстової інформації можуть бути скориговані в процесі роботи за погодженням з керівником.

HUH оформляється згідно СОККР-ІКС-2001

Плановий термін захисту роботи: кінець листопада 2016 року.

Виконавець роботи: _____ Камко П.С.

Керівник роботи: _____ Бивойно П.Г.

Дата видачі завдання: "10" жовтня 2016.

Дата узгодження завдання: "__" _____ 2016.

4.2 Аналіз задачі та діаграма класів

Аналіз опису інформаційної системи, що наведений у технічному завданні свідчіть про те, систему можна представити у вигляді сильно розгалуженого дерева. Представлення такої інформаційної системи у вигляді графа наведено на рисунку 4.1.

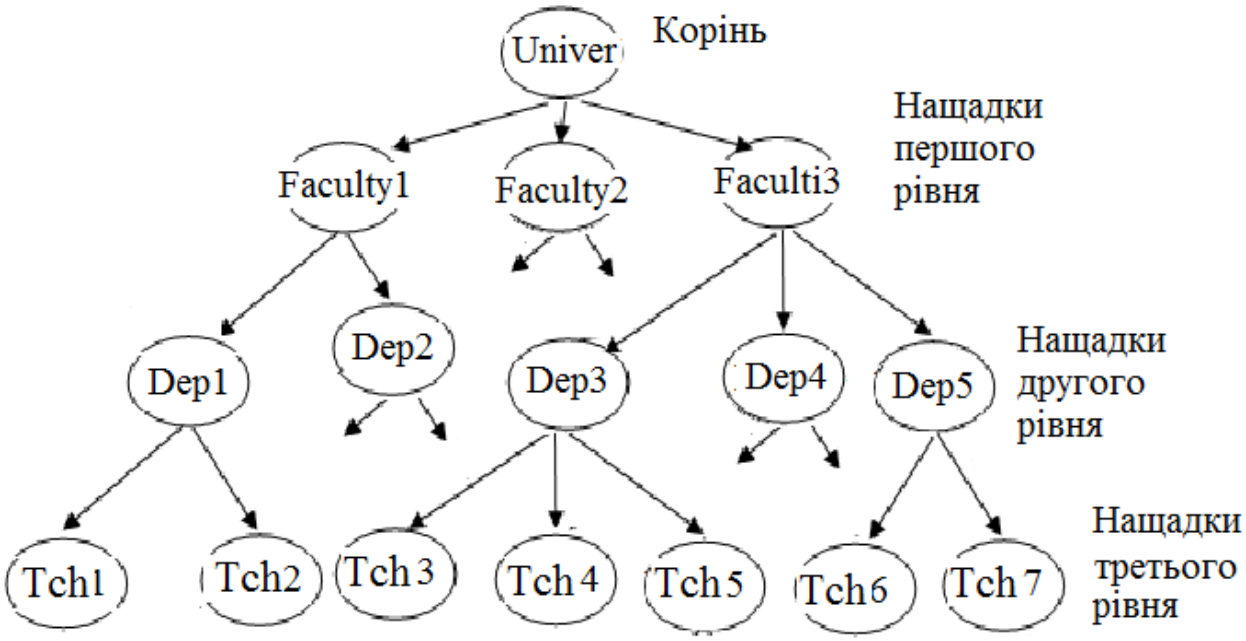

Рисунок 4.1 – Логічна структура інформаційної системи

Така логічна структура може бути реалізована за допомогою об’єктів класу DefaultMutableTreeNode, що реалізує інтерфейс TreeNode. Ці типи визначені у пакеті javax.swing.tree. Інтерфейс TreeNode дозволяє вкладати у такі об’єкти дані будь якого типу, об’єднувати у деревоподібну структуру і відображати у візуальному компоненті типу JTree.

Для реалізації завдання слід перш за все створити класі, які будуть відтворювати абстракції «Університет», «Факультет», «Кафедра» та «Викладач». Окрім того доцільно створити для цих абстракцій узагальнений тип, у якому слід визначити спільні для всіх перелічених абстракцій властивості та поведінку. Наявність такого типу дозволить створювати узагальнені методи роботи з деревом.

Спільним атрибутом для перелічених абстракцій може бути назва (ім’я), а однакова поведінка полягає у відображенні діалогових вікон з інформацією про себе і про сина(найближчого спадкоємця).

На діаграмі класів, рисунок 4.2, узагальненим типом є абстрактний клас AnyNode, а класами, що відтворюють абстракції «Університет», «Факультет», «Кафедра» та «Викладач», є класи Univer, Faculty, Dept, Teacher.

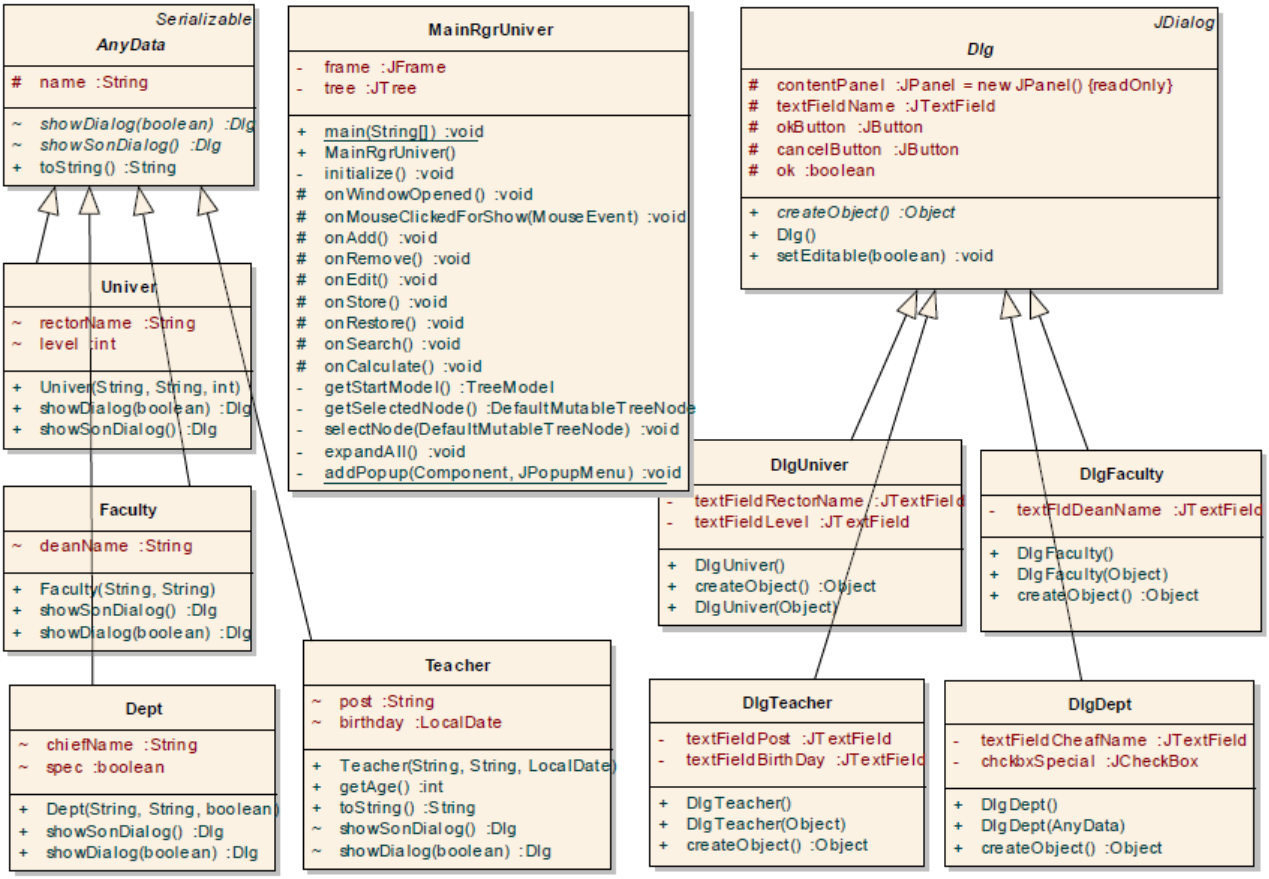

Рисунок 4.2 – Діаграма класів інформаційної системи

Другу ієрархію класів утворюють класи діалогів для відображення інформації про зміст вузлів дерева. Так само як і у попередньому випадку діалоги успадковують абстрактний клас Dlg, куди винесено загальну поведінку та властивості діалогів. На діаграмі класів, рисунок 4.2, класами, що відображують інформацію про «Університет», «Факультет», «Кафедру» та «Викладача», є класи DlgUniver, DlgFaculty, DlgDept, DlgTeacher.

Візуальна частина проекту представлена класом MainRgrUniver

4.3 Реалізація проекту

4.3.1 Реалізація першого варіанту класів для об’єктів дерева

Об’єктами, що містяться у вузлах дерева можуть бути «університет», «факультет», «кафедра» або «викладач». Звичайно, це різні об’єкти, але для того, щоб основні маніпуляції з цими об’єктами у межах нашої структури даних не залежали від їх особливостей, створимо для цих об’єктів загальний суперклас, методи якого визначать спільний інтерфейс для цих об’єктів, а поля будуть містити посилання на однакові властивості.

4.3.1.1 Початковий варіант класу AnyData

– Створімо засобами Eclipse клас AniData, що буде батьківським для класів, що відтворюватимуть «університет», «факультет», «кафедра» або «викладач».

```
public abstract class AnyData {
	protected String name;

	@Override
	public String toString() {
		return name;
	}
}
```

Слід відзначити, що основна цінність цього класу не у тому, що до нього винесено поле name із майбутніх класів спадкоємців, а у тому, що він може використовуватися як тип для об'єктів цих класів. А це дасть змогу писати поліморфний код.

4.3.1.2 Початковий варіант класу Univer

– Створімо засобами Eclipse клас Univer, що відтворює абстракцію «університет».

```
public class Univer extends AnyData {

	String rectorName;
	int level;

	public Univer(String name, String rectorName, int level)
                                                    throws Exception {
		if(level<1 || level>4)
			throw new Exception("Incorrect level");
		this.level = level;
		this.name = name;
		this.rectorName = rectorName;
	}
}
```

4.3.1.3 Початковий варіант класу Univer

– Створімо засобами Eclipse клас Faculty, що відтворює абстракцію «факультет».

```
public class Faculty extends AnyData{
	String deanName;
	public Faculty(String name, String deanName) {
		this.name = name;
		this.deanName = deanName;
	}
}
```

4.3.1.4 Початковий варіант класу Dept

– Створімо засобами Eclipse клас Dept, що відтворює абстракцію «кафедра».

```
public class Dept extends AnyData{
	String chiefName;
	boolean spec; // Чи випускаюча?
	public Dept(String name, String cheafFio, boolean spec) {
		this.name = name;
		this.chiefName = cheafFio;
		this.spec = spec;
	}
}
```

4.3.1.5 Початковий варіант класу Teacher

– Створімо засобами Eclipse клас Teacher, що відтворює абстракцію «викладач».

```
public class Teacher extends AnyData {

	String post;
	LocalDate birthday;

	public Teacher(String name, String post, LocalDate birthday) {
		this.name = name;
		this.post = post;
		this.birthday = birthday;
	}

	// Метод, що повертає вік викладача
	public int getAge() {
		return Period.between(birthday, LocalDate.now()).getYears();
	}

	@Override
	public String toString() {
		return post + " " + name;
	}
}
```

4.3.2 Створення інформаційно-діалогових вікон об’єктів проекту

Для введення та виведення повної інформації про об’єкти нашої системи потрібно створити чотири діалогових вікна («університет», «факультет», «кафедра» та «викладач»), які будуть використовуватися під час роботи з деревом.

Окрім цього, доцільно створити ще один, батьківський діалог, у якому можна реалізувати загальні функції для усіх діалогів. Окрім того, тип цього діалогу може використовуватися для опису усіх діалогів спадкоємців, що дає можливість писати узагальнений код.

4.3.2.1 Створення батьківського діалогу Dlg

Для створення цього діалогу використаймо функцією Swing Designer->JDialog і назвемо клас Dlg.

Виходячи з того, що поле name буде у кожного діалогу спадкоємця, створимо відповідні компоненти у батьківському діалозі.

Додаймо до створеного діалогу компоненти JLabel із текстом Name та компонент JTextField, який назвемо textFieldName.

Зовнішній вигляд діалогу буде таким, як показано на рисунку 4.3

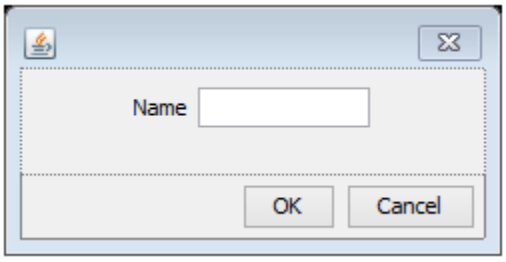

Рисунок 4.3 – Початковий вигляд діалогу Dlg

Далі слід забезпечити доступ до кнопок OK та Cancel кнопки. Для цього можна скористатися функцією “Expose component” контекстного меню кнопок. Як результат, у класі діалогу з’являться поля з посиланнями на кнопки.

Після цього переходимо на закладку Source.

Призначимо специфікатор доступу protected для поля contentPanel, поля textFieldName та полів для посилань на кнопки, бо ці компоненти будуть використовуватися у класах спадкоємцях.

Створімо також поле ok типу boolean зі специфікатором protected, яке буде зберігати інформацію про те, яку кнопку натиснув користувач «OK» чи «Cancel».

Після цього поля класу, що створювалися та змінювалися вручну можуть виглядати так:

```
protected final JPanel contentPanel = new JPanel();
protected JTextField textFieldName;
protected JButton okButton;
protected JButton cancelButton;
protected boolean ok;
```

Далі слід вилучити з класу метод main(), після чого оголосити клас абстрактним.

Після цього можна створити абстрактний метод createObject, що буде у класах спадкоємцях повертати відповідний об’єкт (це буде або «університет», або «факультет», або «кафедра» чи «викладач»). Декларація throws у цьому методі необхідна, бо у його реалізаціях будуть використовуватися конструктори, що викидають вийнятки.

```
public abstract Object createObject() throws Exception;
```

Конструктор класу діалогу слід доповнити викликом методу setModal() з параметром true. Це зробить усі діалоги спадкоємці модальними.

```
public Dlg() {
      setModal(true);
...
```

Створімо ще метод setEditable(), за допомогою якого будемо визначати, чи можна редагувати дані у діалогах.

```
public void setEditable(boolean b) {
      okButton.setVisible(b);
      if(b)
            cancelButton.setText("Cancel");
      else
            cancelButton.setText("Exit");
      for(Component c: contentPanel.getComponents())
            c.setEnabled(b);
}
```

На завершення створимо обробники подій натискання на кнопки «OK» чи «Cancel». Натискання на будь яку з цих кнопок має закривати діалог, а поле ok має приймати значення true тільки у разі натискання на кнопку «OK» .

Таким чином, метод обробки події actionPerformed для кнопки «OK» має виглядати так:

```
public void actionPerformed(ActionEvent e) {
      ok=true;
      setVisible(false);
}
```

А метод обробки події actionPerformed для кнопки «Cancel» буде таким:

```
public void actionPerformed(ActionEvent e) {
      ok=false;
      setVisible(false);
}
```

4.3.2.2 Створення діалогу для даних про університет

Діалог DlgUniver почнемо створювати як звичайний клас (не візуальний), **успадковуючи діалог Dlg та його конструктор і абстрактні методи**.

У конструкторі класу, що було створено, слід викликати метод setBounds, що дозволить налаштовувати розміри діалогового вікна:

```
public DlgUniver() {
      setBounds(100, 100, 100, 100);
}
```

Після цього клас доцільно закрити і відкрити його за допомогою візуального редактора та додати до панелі компоненти типу JLabel та JTextField для введення інформації про ректора та рівень акредитації, рисунок 4.4.

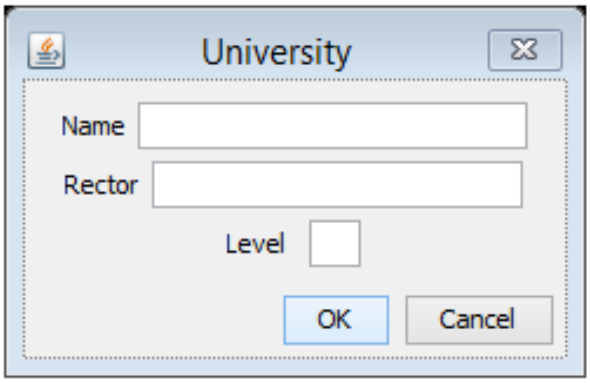

Рисунок 4.4 – Вигляд діалогу DlgUnivers

Тепер можна перейти до редагування коду класу і реалізувати метод createObject:

```
public Object createObject() throws Exception {
      if(!ok) return null;
      String name = textFieldName.getText();
      String fioRector = textFieldRectorName.getText();
      int level = Integer.parseInt(textFieldLevel.getText());
      return new Univer(name, fioRector, level);
}
```

Створений діалог потребує ще одного конструктора, який приймає дані вузла і відображає їх у своїх компонентах. Цей конструктор буде використовуватися у режимах відображення та корегування даних:

```
public DlgUniver(Object data) {
      this();
      Univer u=(Univer) data;
      textFieldName.setText(u.name);
      textFieldRectorName.setText(u.rectorName);
      textFieldLevel.setText(String.valueOf(u.level));
}
```

4.3.2.3 Створення діалогів для факультету, кафедри, викладача

Технологія створення діалогів і структура класів DlgFaculty, DlgDept, DlgTeacher такі самі як і у попередньому випадку. Тому тут ми наводити їх не будемо.

4.3.3 Доопрацювання класів для об’єктів дерева

Після того, як створено діалоги, можна пов’язати їх з відповідними класами об’єктів дерева, бо саме через ці об’єкти будуть викликатися діалоги.

4.3.3.1 Доопрацювання класу AnyData

У цьому класі визначимо абстрактні методи showDialog та showSonDialog.

Перший з цих методів має повернути і активізувати діалог для роботи з даними вузла. Він може бути налаштований або на перегляд, або на введення та редагування інформації. Режим буде залежати від значення параметру b.

```
abstract Dlg showDialog(boolean b);
```

Другий метод має повертати і активізувати діалог для роботи з даними спадкоємця. Цей метод буде потрібен у режимі додавання вузлів до дерева.

```
abstract Dlg showSonDialog();
```

4.3.3.2 Доопрацювання класу Univer

Доопрацювання класу полягає у реалізації абстрактних методів суперкласу

```
@Override
public Dlg showDialog( boolean b) {
      DlgUniver d = new DlgUniver(this);
      d.setEditable(b);
      d.setVisible(true);
      return d;
}
@Override
public Dlg showSonDialog() {
      DlgFaculty d = new DlgFaculty();
      d.setVisible(true);
      return d;
}
```

4.3.3.3 Доопрацювання класів Faculty, Dept, Teacher

Доопрацювання цих класу також потребує реалізації абстрактних методів суперкласу, які можуть бути реалізовані за такою ж схемою, як і для попереднього класу.

Особливість реалізації методу showSonDialog у класі Teacher полягає у тому, що він має повертати null. Це пов’язано з тим, що для цього класу спадкоємці не передбачені завданням.

4.3.4 Реалізація візуальної частини

4.3.4.1 Дизайн візуальної частини

На рисунку 4.5 зображено вигляд візуальної частини додатку. Для відображення дерева використовується компонент класу JTree. Група кнопок для роботи з деревом розташована на окремій панелі.

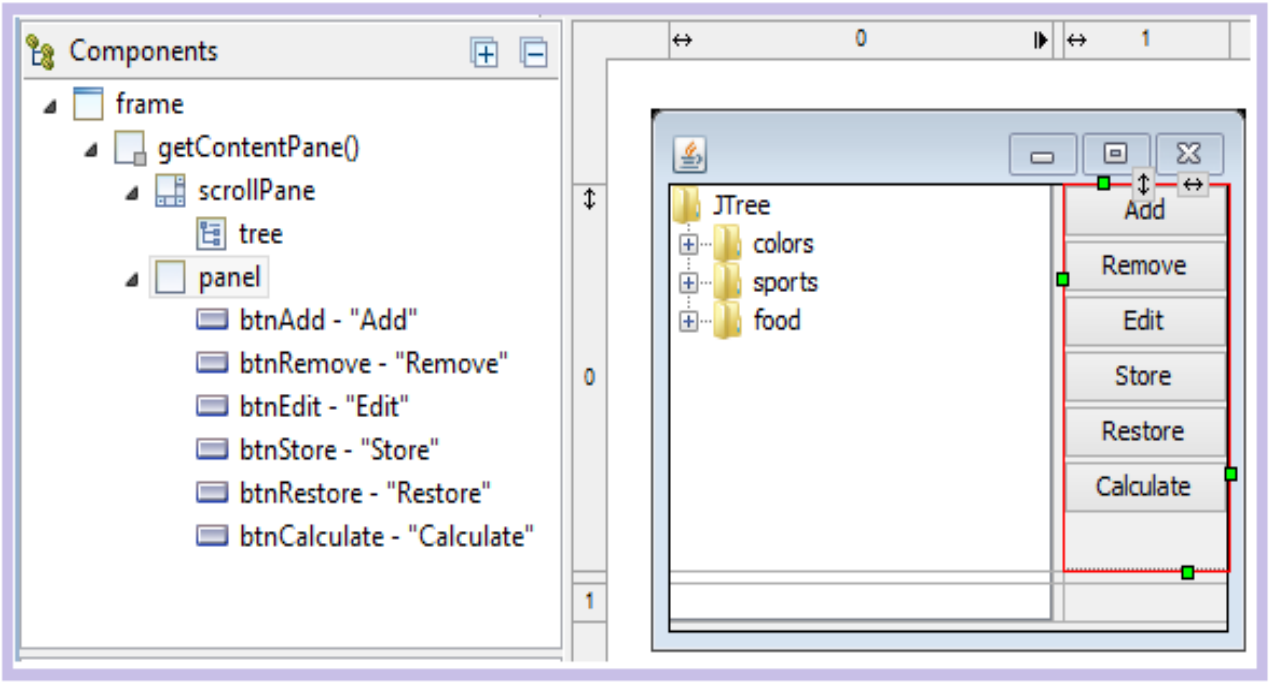

Рисунок 4.5 – Візуальна частина на етапі розробки

4.3.4.2 Створення моделі для компоненту JTree

Перш ніж створювати модель, слід зробити доступним компонент класу JTree, який буде відображати дерево. Для цього у режимі дизайну слід викликати для нього функцію “Expose component”. Внаслідок цього у класі з’явиться поле tree типу JTree.

Об’єкт tree використовується тільки для відображення дерева, а структура дерева і його дані визначаються моделлю, яка передається об’єкту tree через метод setModel.

Конструктори класу JTree надають можливість використовувати в якості параметру не модель, а об’єкт типу TreeNode, яки має бути коренем дерева. Інтерфейс TreeNode визначає перелік методів вузла дерева, які необхідні JTree для побудови дерева. В пакеті javax.swing.tree є клас DefaultMutableTreeNode, який реалізує інтерфейс TreeNode.

Саме цей клас ми будемо використовувати для створення вузлів дерева.

Після цього, за допомогою методу add об’єднаймо створені вузли у ланцюжок, взявши вузол з університетом у якості кореню.

Далі створимо новий об’єкт JTree, передавши корінь у якості параметру конструктора. Як результат, у об’єкті класу JTree буде створено модель дерева, яку ми можемо отримати за допомогою методу getModel().

Нижче наведено текст методу, що виконує усі перелічені операції.

```
protected TreeModel getStartModel() throws Exception {
      // Створюємо об'єкти для вузлів дерева
      Univer u = new Univer("Чернігів НТУ", "Шкарлет С.М.", 4);
      Faculty f = new Faculty("ФЕІТ", "Іванець С.А.");
      Dept d = new Dept("Кафедра ПІ", "Литвинов В.В", true);
      Teacher t = new Teacher("Скітер І.С.", "доцент",
                        LocalDate.of(1976,Month.JUNE, 05));
      // Створюємо вузли дерева
      DefaultMutableTreeNode root = new DefaultMutableTreeNode(u);
      DefaultMutableTreeNode fNod = new DefaultMutableTreeNode(f);
      DefaultMutableTreeNode dNod = new DefaultMutableTreeNode(d);
      DefaultMutableTreeNode tNod = new DefaultMutableTreeNode(t);
      // Зв'язуємо вузли між собою
      root.add(fNod); fNod.add(dNod); dNod.add(tNod);
      // Отримуємо та повертаємо створену модель
      return (new JTree(root)).getModel();
}
```

Для того, щоб модель, що створюється у методі getStartModel(), відображалася після запуску нашого додатку, потрібно з подією windowOpened вікна додатку пов’язати такий метод:

```
protected void onWindowOpened() {
      try {
            tree.setModel(getStartModel());
            // Розкриваємо усі вузли
            for (int i = 0; i < tree.getRowCount(); i++)
                  tree.expandRow(i);
      } catch (Exception e1) {
            e1.printStackTrace();
}
                                                                  }
```

Після цього можна перевірити працездатність створених класів. Після запуску додатку на екрані має з’явитися вікно, що зображено на рисунку 2.2.

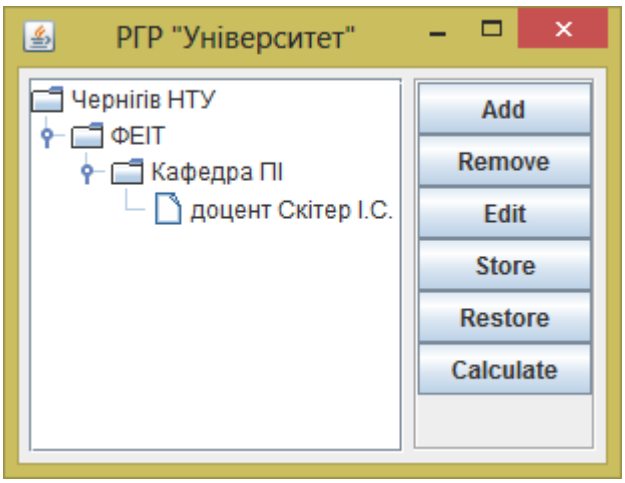

Рисунок 4.6 – Вікно проекту із стартовою моделлю

4.3.4.3 Створення допоміжних методів

Використання допоміжних методів, що наведені нижче спрощує реалізацію основних методів роботи з деревом.

```
// Попередня обробка посилання на вибраний вузол
private DefaultMutableTreeNode getSelectedNode() {
	// Отримуємо посилання на вибраний вузол дерева
	// Якщо вузол не вибрано, посилання дорівнює null
	Object selectNode = tree.getLastSelectedPathComponent();
	if (selectNode == null) {
		JOptionPane.showMessageDialog(tree,
					"Node was not selected",
					frame.getTitle(),
					JOptionPane.ERROR_MESSAGE);
		return null;
	}
	// Приводимо посилання до типу вузла
	return (DefaultMutableTreeNode) selectNode;
}

// Як розгорнути усі вузли
private void expandAll() {
	for (int i = 0; i < tree.getRowCount(); i++) {
		tree.expandRow(i);
	}
}


// Як виділити вузол
private void selectNode(DefaultMutableTreeNode node) {
	int n = 0;
	DefaultMutableTreeNode root =
		(DefaultMutableTreeNode) tree.getModel().getRoot();
	Enumeration<DefaultMutableTreeNode> enm = root.children();
	while (enm.hasMoreElements()) {
		DefaultMutableTreeNode nod = enm.nextElement();
		if (nod == node) {
			tree.setSelectionRow(n);
			return;
		}
		n++;
	}
}
```

4.3.4.4 Реалізація перегляду змісту вузлів

Для реалізації перегляду змісту вузла, який було виділено, будемо використовувати потрійний клік правою кнопкою миші по компоненту JTree.

Створімо обробника події mouseClicked для компонента JTree з такою реалізацією методу обробки події:

```
tree.addMouseListener(new MouseAdapter() {
	@Override
	public void mouseClicked(MouseEvent e) {
		onMouseClicked(e);
	}
});
```

Після цього реалізуємо метод onMouseClick. Створюючи цей метод, ми будемо використовувати узагальнені типи для даних вузла та діалогів. Дані будуть проходити під типом AnyData, а діалоги під типом Dlg. Це надає змогу написати узагальнений метод, який буде працювати на усіх рівнях, незалежно від класів до яких належать дані та діалоги.

```
private void onMouseClicked(MouseEvent e) {
	if (e.getClickCount() != 3 ||
			e.getButton() != MouseEvent.BUTTON3)
		return;
	DefaultMutableTreeNode node = getSelectedNode();
	if (node == null)
		return;
	AnyData data = (AnyData) node.getUserObject();
	Dlg dlg = data.showDialog(false);
	((JDialog) dlg).dispose();
}
```

Тепер можна протестувати не тільки наведений метод, але й роботу діалогів та даних вузлів у режимі перегляду.

4.3.4.5 Реалізація операції додавання вузла

Створімо обробника події натискання на кнопку «Add» і реалізуємо метод обробки події через виклик приватного методу onAdd():

```
btnAdd.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent e) {
		onAdd();
	}
});
```

Якщо використати лямбда функцію, наведений код може виглядати простіше:

```
btnAdd.addActionListener(e->onAdd());
```

Створюючи метод onAdd(), ми також будемо використовувати узагальнені типи для даних вузла та діалогів. Це надає можливість написати

узагальнений метод, який буде працювати на усіх рівнях, незалежно від класів до яких належать дані та діалоги.

Окрім того, у цьому методі обробляються винятки, які можуть викидати конструктори у разі некоректних даних, що введені через діалог.

Сам метод onAdd() виглядає так:

```
protected void onAdd() {
	// Отримуємо посилання на вибраний вузол дерева
	DefaultMutableTreeNode parent = getSelectedNode();
	if (parent == null) return;
	//Через дані вузла, викликаємо діалог спадкоєця
	AnyData parentData = (AnyData) parent.getUserObject();
	Dlg dlg = parentData.showSonDialog();
	if (dlg == null) return;
	// Отримуємо об'єкт даних від діалога
	Object obj;
	try {
		obj = dlg.createObject();
	} catch (Exception e) {
		JOptionPane.showMessageDialog(tree, e.getMessage(),
				frame.getTitle(), JOptionPane.ERROR_MESSAGE);
		return;
	}
	((JDialog) dlg).dispose();
	if (obj == null)
		return;
	//Створюємо новий вузол з даними
	DefaultMutableTreeNode newSon =
				new DefaultMutableTreeNode(obj);
	// Додаємо вузол
	parent.add(newSon);
	// Оновлюємо вид дерева і розкриваємо усі вузли
	tree.updateUI();
	expandAll();
}
```

4.3.4.6 Реалізація операції видалення вузла

Для початку слід створити обробника події натискання на кнопку Remove, який буде викликати метод onRemove(), так само як і у попередньому пункті.

Далі слід реалізувати метод onRemove(), використовуючи узагальнені типи для даних вузла та діалогів.

```
private void onRemove() {
// Отримуємо посилання на вибраний вузол дерева
	DefaultMutableTreeNode node = getSelectedNode();
	if (node == null)
		return;
	// Видаляємо вузол
	node.removeFromParent();
	//Звільняємо посилання на вибраний вузол дерева
	tree.setSelectionPath(null);
	tree.updateUI();
}
```

4.3.4.7 Реалізація операції редагування даних вузла

Так само як і у попередньому пункті, для початку слід створити обробника події натискання на кнопку Edit, який буде викликати метод onEdit().

Далі слід реалізувати цей метод, який завдяки поліморфізму буде працювати з будь яким вузлом:

```
private void onEdit() {
	// Отримуємо посилання на вибраний вузол дерева
	DefaultMutableTreeNode node = getSelectedNode();
	if (node == null) return;
	AnyData data = (AnyData) node.getUserObject();
	// Надаємо дані вузла для редагування
	Dlg dlg = data.showDialog(true);
	//Отримуємо відредаговані дані
	Object obj;
	try {
		obj = dlg.createObject();
	} catch (Exception e) {
		JOptionPane.showMessageDialog(tree, e.getMessage(),
				frame.getTitle(), JOptionPane.ERROR_MESSAGE);
		return;
	}
	((JDialog) dlg).dispose();
	if (obj == null) return;
	node.setUserObject(obj);
	// Оновлюємо вигляд дерева
	tree.updateUI();
}
```

4.3.4.8 Реалізація операції збереження дерева у файлі

Для виконання цієї операції з кнопкою «Store» пов’яжемо метод onStore:

```
private void onStore() {
	if (tree.getModel() == null)
		return;
	JFileChooser fileChooser =
		new JFileChooser("Серіалізация моделі дерева");
	fileChooser.showSaveDialog(frame);
	try {
		File f = fileChooser.getSelectedFile();
		String fName = f.getAbsolutePath();
		FileOutputStream fileStream =
			new FileOutputStream(fName);
		ObjectOutputStream out =
			new ObjectOutputStream(fileStream);

		// серіализуємо об'єкт
		out.writeObject(tree.getModel());
		out.close(); // закриваємо потік
	} catch (Exception e1) {
		JOptionPane.showMessageDialog(tree,
			"Помилка відкриття файлу",
			"Збереження дерева у файлі",
			JOptionPane.ERROR_MESSAGE);
		return;
	}
	tree.setModel((new JTree()).getModel());
}
```

4.3.4.9 Реалізація відновлення дерева з файлу

Для виконання цієї операції з кнопкою «Restore» пов’яжемо метод onRestore:

```
private void onRestore() {
	FileDialog fileDialog = new FileDialog(frame);
	fileDialog.setMode(FileDialog.LOAD);
	fileDialog.setVisible(true);
	String dr = fileDialog.getDirectory();
	String fn = fileDialog.getFile();
	if (dr == null || fn == null)
		return;
	String fName = dr + fn;
	try {
		// створюємо потік для десеріалізації моделі дерева
		ObjectInputStream in = new ObjectInputStream(
				new FileInputStream(fName));
		// десеріалізація
		TreeModel model = (TreeModel) in.readObject();
		// Передаємо модель дереву
		tree.setModel(model);
		in.close(); // закриємо потік
```

```
        } catch (Exception e1) {
            JOptionPane.showMessageDialog(tree,
                    "Помилка десеріалізації дерева",
                    "Десеріалізація", JOptionPane.ERROR_MESSAGE);
            return;
        }
        expandAll();
    }
```

4.3.4.10 Реалізація функцій пошуку та підрахунків для дерева

Для реалізації цих функцій на візуальній частині передбачена кнопка «Find/Calculate».

Пов’яжемо з цією кнопкою компонент JPopupMenu і додамо до нього два компоненти типу JMenuItem, так як це показано на рисунку 2.3.

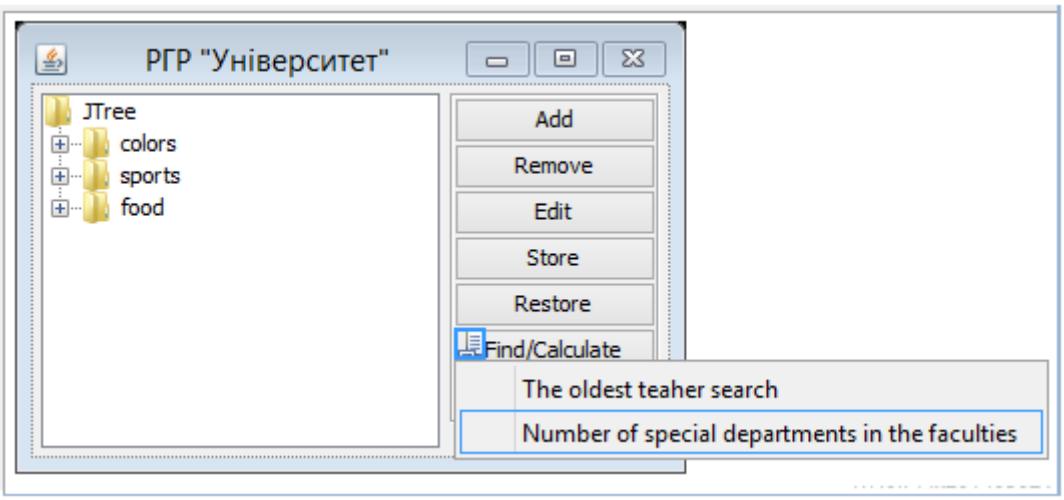

Рисунок 4.7 – Меню для кнопки Find/Calculate

У результаті виконання цих операцій окрім компонентів меню з’являється приватний статичний метод addPoup, у якому створюється об’єкт абстрактного класу MouseAdapter, що виступає у ролі обробника подій миші.

```
    private static void addPopup(Component component,
                                 final JPopupMenu popup) {
        component.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                // if (e.isPopupTrigger())
                showMenu(e);
            }
            // public void mouseReleased(MouseEvent e) {
            //     if (e.isPopupTrigger())
            //     showMenu(e);
            // }
            private void showMenu(MouseEvent e) {
                popup.show(e.getComponent(), e.getX(), e.getY());
            }
        });
    }
```

При цьому перевизначаються методи реакції на натискання та відпускання клавіші миші через виклик методу showMenu, створеного тут же.

Методи реакції на клавіші миші реалізують стандартний виклик меню через праву кнопку за допомогою перевірки умови «if (e.isPopupTrigger())».

Якщо цю перевірку вилучити, меню буде викликатися і при натисканні на ліву кнопку миші. Саме тому частина коду у методі addPopup, що наведено вище, закоментовано.

4.3.4.11 Пошук викладача, найстаршого за віком

Для реалізації пошуку пов'яжемо з елементом меню «The oldest teacher search» метод onSearch:

```
protected void onSearch() {
	// Отримуємо посилання на вибраний вузол дерева
	DefaultMutableTreeNode node = getSelectedNode();
	if (node == null)
		return;
	int maxAge = 0;
	DefaultMutableTreeNode oldest = null;
	// Отримуємо ітератор для вузлів дерева
	Enumeration<DefaultMutableTreeNode> enm =
				node.postorderEnumeration();
	while (enm.hasMoreElements()) {
		DefaultMutableTreeNode currentNode = enm.nextElement();
		Object data = currentNode.getUserObject();
		if (!(data instanceof Teacher))
			continue;
		int age = ((Teacher) data).getAge();
		if (age > maxAge) {
			maxAge = age;
			oldest = currentNode;
		}
	}
	System.out.println(oldest);
	selectNode(oldest);
	((AnyData) oldest.getUserObject()).showDialog(false);
}
```

4.3.4.12 Підрахунок кількості випускаючих кафедр на факультетах

Для реалізації пошуку пов'яжемо з елементом меню «The oldest teacher search» метод onSearch:

```
private void onCalculate() {
	DefaultMutableTreeNode root =
			(DefaultMutableTreeNode) tree.getModel()
			.getRoot();
```

```
        // Ітератор для факультетів
        Enumeration<DefaultMutableTreeNode> enmFclt =
                    root.children();
        while (enmFclt.hasMoreElements()) {
              DefaultMutableTreeNode fclt = enmFclt.nextElement();
              int nSpec = 0;
              // Ітератор для кафедр
              Enumeration<DefaultMutableTreeNode> enmDep =
                    fclt.children();
              while (enmDep.hasMoreElements()) {
                    DefaultMutableTreeNode dep = enmDep.nextElement();
                    Object data = dep.getUserObject();
                    if (((Dept) data).spec) {
                          nSpec++;
                    }
              }
              // Результат по факультету
              System.out.println(
                    fclt + " has " + nSpec + " specdepartments");
        }
  }
```

Рекомендована література

1. Бадд Т. Объектно-ориентированное программирование в действии. Санкт-Петербург: Питер, 1997.
2. Буч Г. Объектно-ориентированное проектирование с примерами применения: Пер. с англ. – М.:Кокорд,1992. – 519с
3. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С++: Пер. с англ. – М.:Диалект,1999
4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. – СПб.:Питер,2001
5. Фаулер Мартин Архитектура корпоративных программных приложений. СПб.: Издательский дом «Вильямс», 2007. – 544с
6. Simon Kendal. Object oriented programming using Java. Ventus Publishing ApS-, 2009. – 209 с.
7. http://omg.org Object Management Group