**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

Чернігівський національний технологічний університет

РЕАЛІЗАЦІЯ  
ПОЛІМОРФНОЇ МОДЕЛІ ДЕРЕВА  
ДЛЯ БАГАТОРІВНЕВОЇ СТРУКТУРИ ДАНИХ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання РГР з дисципліни

« Об’єктно-орієнтоване програмування»

для студентів напряму підготовки

6.050103 –“Програмна інженерія”

ЗАТВЕРДЖЕНО

на засіданні кафедри

інформаційних та комп’ютерних систем

протокол № 1 від 31.08.17

**Чернігів ЧНТУ 2017**

Реалізація поліморфної моделі дерева для багаторівневої структури даних. Методичні вказівки до виконання РГР з дисципліни «Об’єктно-орієнтоване програмування» для студентів напрямів підготовки 6.050103 – „Програмна інженерія”. /Укл.: Бивойно П.Г., Бивойно Т.П. – Чернігів: ЧНТУ, 2017. – 60 с.

Укладачі: Бивойно Павло Георгійович, доцент, к.т.н.,

Бивойно Тарас Павлович, старший викладач

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

Рецензент: С. О. Нестеренко, канд. техн. наук, доцент кафедри інформаційних і комп'ютерних систем Чернігівського державного технологічного університету

ЗМІСТ

[Вступ 6](#_Toc467236437)

[1 ЗАВДАННЯ ДО РОЗРАХУНКОВО - ГРАФІЧНОЇ РОБОТИ 7](#_Toc467236438)

[2 ВИМОГИ ДО РОБОТИ 8](#_Toc467236439)

[2.1 Загальні вимоги 8](#_Toc467236440)

[2.2 Обов’язкові розділи пояснювальної записки 8](#_Toc467236441)

[2.2.1 Технічне завдання 8](#_Toc467236442)

[2.2.2 Аналіз та розробка системи що підлягає реалізації 9](#_Toc467236443)

[2.2.3 Реалізація моделі дерева 9](#_Toc467236444)

[2.2.4 Реалізація моделі предметної області 9](#_Toc467236445)

[2.2.5 Реалізація шару подання 9](#_Toc467236446)

[2.2.6 Тестування системи 9](#_Toc467236447)

[3 ТЕОРЕТИЧНІ ВІДОМОСТІ 10](#_Toc467236448)

[3.1 Логічна структура сильно розгалуженого дерева 10](#_Toc467236449)

[3.2 Аналіз структур даних, що можна використовувати для збереження нащадків вузла дерева 10](#_Toc467236450)

[3.2.1 Посилальні структури даних 10](#_Toc467236451)

[3.2.1.1 Однонаправлений список 11](#_Toc467236452)

[3.2.1.2 Двонаправлений список 11](#_Toc467236453)

[3.2.1.3 Бінарне дерево 11](#_Toc467236454)

[3.2.2 Структури, що використовують масиви 12](#_Toc467236455)

[3.2.2.1 Звичайний масив вказівників 12](#_Toc467236456)

[3.2.2.2 Циклічний масив 13](#_Toc467236457)

[3.2.2.3 Бінарна купа 13](#_Toc467236458)

[3.2.3 Комбіновані структури 14](#_Toc467236459)

[3.2.3.1 Хеш-таблиці 14](#_Toc467236460)

[3.2.3.2 Префіксні дерева 14](#_Toc467236461)

[3.3 Засоби Java для реалізації шару подання 15](#_Toc467236462)

[3.3.1 Засоби Java для створення та відображення дерева 15](#_Toc467236463)

[3.3.1.1 Клас javax.swing.JTree 15](#_Toc467236464)

[3.3.1.2 Інтерфейс javax.swing.tree.TreeModel 16](#_Toc467236465)

[3.3.1.3 Інтерфейс javax.swing.tree.TreeNode 16](#_Toc467236466)

[3.3.1.4 Інтерфейс javax.swing.tree.MutableTreeNode 17](#_Toc467236467)

[3.3.2 Засоби для введення/виведення інформації про об’єкти предметної області 17](#_Toc467236468)

[3.3.3 Засоби Java для виклику операцій користувача над деревом 18](#_Toc467236469)

[3.3.3.1 Кнопки 18](#_Toc467236470)

[3.3.3.2 Радіоклавіши 19](#_Toc467236471)

[3.3.3.3 Кнопки меню 19](#_Toc467236472)

[3.3.3.4 Обробка події натискання на кнопку 19](#_Toc467236473)

[4 ПРИКЛАД ПОБУДОВИ СИСТЕМИ 22](#_Toc467236474)

[4.1 Аналіз задачі та проектування системи 22](#_Toc467236475)

[4.1.1 Модель предметної області 22](#_Toc467236476)

[4.1.2 Інтерфейс користувача 23](#_Toc467236477)

[4.1.3 Модель дерева 24](#_Toc467236478)

[4.2 Структура пакетів проекту 26](#_Toc467236479)

[4.3 Реалізація класів для об’єктів предметної області 26](#_Toc467236480)

[4.3.1 Інтерфейс IUserData 26](#_Toc467236481)

[4.3.2 Клас UniverData 26](#_Toc467236482)

[4.3.3 Клас FacultyData 27](#_Toc467236483)

[4.3.4 Клас DeptData 28](#_Toc467236484)

[4.3.5 Клас TeacherData 29](#_Toc467236485)

[4.4 Дизайн головного вікна 29](#_Toc467236486)

[4.4.1 Використання об’єктів Action для меню та кнопок 30](#_Toc467236487)

[4.4.1.1 Створення об’єктів типу Action 30](#_Toc467236488)

[4.4.1.2 Зв’язування об’єктів типу Action з кнопками 31](#_Toc467236489)

[4.4.1.3 Зв’язування об’єктів типу Action з меню 32](#_Toc467236490)

[4.5 Створення класів панелей для компоненту JTabbedPane 32](#_Toc467236491)

[4.5.1 Панель заставки 33](#_Toc467236492)

[4.5.1.1 Клас ImagePanel 33](#_Toc467236493)

[4.5.1.2 Тестування класу ImagePanel 34](#_Toc467236494)

[4.5.2 Панель для виведення результатів запитів 35](#_Toc467236495)

[4.5.2.1 Клас QueryPanel 35](#_Toc467236496)

[4.5.2.2 Тестування класу QueryPanel 36](#_Toc467236497)

[4.5.3 Панелі введення/виведення 36](#_Toc467236498)

[4.5.3.1 Інтерфейс IDataPanel для панелей введення/виведення 36](#_Toc467236499)

[4.5.3.2 Клас UniverPanel для введення даних про університет 37](#_Toc467236500)

[4.5.3.3 Тестування класу UniverData 38](#_Toc467236501)

[4.5.3.4 Клас FacultyPanel для роботи з даними про факультет 39](#_Toc467236502)

[4.5.3.5 Тестування панелі FacultyPanel 39](#_Toc467236503)

[4.5.3.6 Клас DeptPanel для роботи з даними про кафедру 39](#_Toc467236504)

[4.5.3.7 Тестування панелі DeptPanel 40](#_Toc467236505)

[4.5.3.8 Клас DeptPanel для роботи з даними про кафедру 40](#_Toc467236506)

[4.5.3.9 Тестування панелі TeacherPanel 41](#_Toc467236507)

[4.6 Реалізація класів для вузлів дерева 41](#_Toc467236508)

[4.6.1 Інтерфейс для вузлів дерева 41](#_Toc467236509)

[4.6.2 Узагальнений клас для вузлів дерева 42](#_Toc467236510)

[4.6.2.1 Реалізація інтерфейсу Comparable 42](#_Toc467236511)

[4.6.2.2 Реалізація інтерфейсу Iterable 43](#_Toc467236512)

[4.6.2.3 Реалізація інтерфейсу TreeNode 43](#_Toc467236513)

[4.6.2.4 Реалізація методів інтерфейсу для роботи з даними вузла 44](#_Toc467236514)

[4.6.2.5 Реалізація методу subTreeIterator 45](#_Toc467236515)

[4.6.3 Класу TeacherNode для збереження даних про викладачів 46](#_Toc467236516)

[4.6.4 Реалізація класу DeptNode 48](#_Toc467236517)

[4.6.5 Реалізація класу FacultyNode 50](#_Toc467236518)

[4.6.6 Клас UniverNode для кореневого вузла 51](#_Toc467236519)

[4.6.7 Реалізація моделі дерева 52](#_Toc467236520)

[4.7 Програмування візуальної частин 53](#_Toc467236521)

[4.7.1 Реалізація методу onCreateRoot() 53](#_Toc467236522)

[4.7.2 Реалізація методу onAdd() 54](#_Toc467236523)

[4.7.3 Реалізація операції видалення вузла 56](#_Toc467236524)

[4.7.4 Реалізація операції редагування даних вузла 56](#_Toc467236525)

[4.7.5 Реалізація перегляду змісту вузлів 57](#_Toc467236526)

[4.7.6 Реалізація операції збереження дерева у файлі 57](#_Toc467236527)

[4.7.7 Реалізація відновлення дерева з файлу 58](#_Toc467236528)

[4.7.8 Пошук викладача, найстаршого за віком 58](#_Toc467236529)

[4.7.9 Підрахунок кількості випускаючих кафедр на факультетах 59](#_Toc467236530)

[Рекомендована література 60](#_Toc467236531)

Вступ

У курсі «Об’єктно-орієнтоване програмування» виконання розрахунково графічної роботи є частиною самостійної роботи студентів над дисципліною.

Головна мета даної роботи полягає у формуванні навичок використання основних концепцій об’єктно орієнтованого програмування для написання узагальненого коду, що може працювати незалежно від структур даних, які використовуються для збереження даних предметної області. Для цього студенти мають створити поліморфну модель сильно розгалуженого дерева, яка передбачає використання для кожного рівня своєї специфічної структури даних для збереження інформації про нащадків.

Окрім того робота передбачає проведення об’єктно-орієнтованого аналізу заданої предметної області, визначення переліку інтерфейсів та класів Java, що можуть бути використані або безпосередньо, або як батьківські класи для реалізації класів майбутнього додатку і, нарешті, подальшу програмну реалізацію цього додатку, що буде вирішувати деякі проблеми заданої предметної області. У складі програмної реалізації має бути візуальна композиція для реалізації графічного інтерфейсу користувача.

Результатом роботи є звіт обсягом приблизно 30 сторінок друкованого тексту оформленого відповідно до стандартів кафедри. Бали за роботу виставляються з урахуванням своєчасності та якості виконання.

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

# ЗАВДАННЯ ДО 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 |

# ВИМОГИ ДО РОБОТИ

## Загальні вимоги

Робота передбачає створення динамічної інформаційної структури даних у вигляді багаторівневого розгалуженого дерева. Студент має самостійно розробити модель для дерева, що реалізує інтерфейси TreeModel та TreeNode з бібліотеки Java, та візуального додатку на основі компоненту JTree та діалогів JDialog, 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.

## 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.

### 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.

### Аналіз та проектування системи, що підлягає реалізації

У цьому розділі слід навести:

* схематичне зображення предметної області у вигляді розгалуженого дерева;
* модель предметної області у вигляді діаграми класів та їх описом;
* проект інтерфейсу користувача з обґрунтуванням вибору компонентів та діаграмою класів самостійно створених компонентів;
* модель деревовидної структури для збереження даних користувача з характеристикою і обґрунтуванням вибраних структур даних на різних рівнях та діаграмою класів.

### Реалізація системи

У цьому розділі слід навести тексти класів, що було створено.

Класи мають бути згруповані по пакетах. Кожний пакет має бути представлений окремим підрозділом.

Спочатку мають бути описані інтерфейси пакету потім класи.

До класів візуальних компонентів слід додавати графічні зображення.

Кожний інтерфейс і клас мають бути представлені окремим пунктом.

Кожний клас і метод мають бути прокоментовані у стилі java.doc.

### Тестування системи

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

Цей підрозділ має слугувати доказом працездатності створеного додатку.

# ТЕОРЕТИЧНІ ВІДОМОСТІ

## Логічна структура сильно розгалуженого дерева

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 3.1.

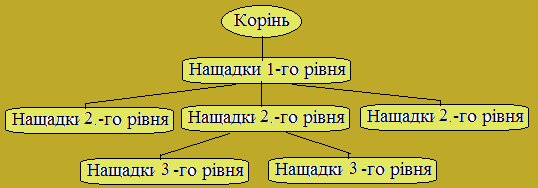


Рисунок 3.1 – Логічна структура інформаційної системи

У сильно розгалуженого дерева кожний вузол може мати довільну кількість нащадків. Це призводить до того, що вузол не може посилатися на кожного нащадку окремо. Можна посилатися тільки на структуру даних, у якій зберігаються нащадки. Кожна з існуючих структур має свої переваги і недоліки. Вибирати структуру потрібно виходячи із переліку операцій, які потрібно виконувати над даними.

## Аналіз структур даних, що можна використовувати для збереження нащадків вузла дерева

Найпростіший спосіб створення структури даних для збереження нащадків вузла – це використання об’єктів типу Collection. Цей інтерфейс реалізовано у низці класів, що використовують різноманітні структури даних. Потрібну реалізацію можна вибирати відповідно до завдань, що вирішує той чи інший рівень дерева. Завдяки єдиному інтерфейсові код обробки вузлів не буде залежати від вибору його реалізації.

Але в даній роботі об’єкт типу Collection будемо використовувати тільки на одному рівні. На інших двох рівнях студент має сам реалізувати структури даних відповідно до завдання.

Коротко розглянемо особливості таких структур.

### Посилальні структури даних

Особливість таких структур полягає у тому, що кожний елемент структури окрім посилання на дані користувача містить посилання на інші елементи структури.

Переваги таких структур у тому, що не використовується зайва пам’ять, бо елементи створюються динамічно у разі появи нових синів. Так само пам’ять вивільняється одразу після видалення елементу. Але процедури виділення пам’яті та створення нового елементу потребують деякого часу

#### Однонаправлений список

В основі такої структури лежить об’єкт, що має посилання на дані користувача (userObject), посилання на наступний вузол у списку (brother) та на батьківський вузол (parent), рисунок 3.2.

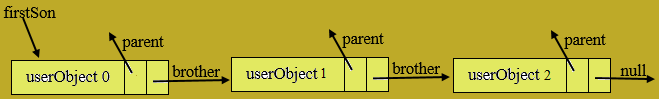


Рисунок 3.2 – Структура однонаправленого списку

Батьківський вузол має посилання firstSon на перший елемент списку.

Головний недолік полягає в тому, що для пошуку потрібного елементу доводиться переглядати увесь список.

Додавання у середину та кінець ускладняється тим, що треба запам’ятовувати інформацію про попередній елемент.

Така структура найкраще підходить для організації стеку.

#### Двонаправлений список

Такий список відрізняється від попереднього тільки тим, що кожний елемент зберігає посилання не тільки на наступного, але й на попереднього елементу, рисунок 3.3. Це спрощує процедури вставки елементів.

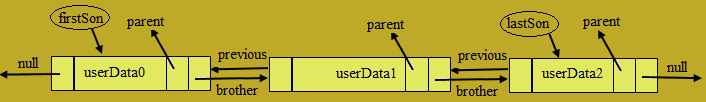


Рисунок 3.3 – Структура двонаправленого списку

Батьківський вузол містить два посилання на таку структуру. Окрім посилання firstSon є ще посилання на кінець списку – lastSon. Це спрощує доступ не тільки до початку, але й до кінця списку.

#### Бінарне дерево

Головна особливість бінарного дерева полягає у тому, що кожен вузол містить посилання на лівого (left) та правого (right) нащадку. Окрім того кожен вузол може містити посилання на вузол вищого рівня(upper). Структура бінарного дерева наведена на рисунку 3.4.

Батьківський вузол містить посилання на корінь дерева (root). А корінь містить посилання на батьківський вузол (parent). Алгоритм додавання елементів до дерева вимагає, щоб елементи, що менші за елемент вищого рівня, розташовувалися ліворуч від нього, а більші – праворуч. Це дозволяє швидко знаходити елементи і обходити дерево у відсортованому порядку. Але для того, щоб дерево не перетворилося у лінійний список, або щось наближене до нього, потрібні процедури балансування дерева.

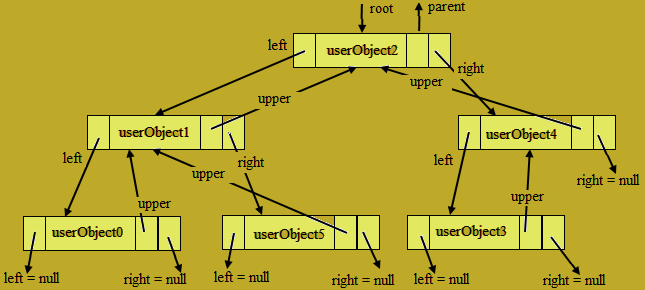


Рисунок 3.4 – Структура бінарного дерева

Зберігати нащадків у бінарному дереві доцільно у тих випадках, коли потрібно впорядковувати перелік синів. Окрім того, завдяки впорядкованості елементів, пошук у дереві відбувається достатньо швидко.

### Структури, що використовують масиви

Перевага масивів перед посилальними структурами полягає у тому, що доступ до елементів за індексом відбувається дуже швидко. Окрім того, елементи масиву містять тільки посилання на дані користувача. Ніяких додаткових посилань нема, а це зменшує обсяг необхідної пам’яті.

З іншого боку, розмір масиву зазвичай перевищує кількість елементів у ньому, бо створювати новий масив перед кожним додаванням нового елементу недоцільно. Але операції збільшення розміру масиву періодично доводиться виконувати і на це витрачається час.

Батьківський вузол у цьому випадку має тримати посилання на масив синів. А сама структура має пам’ятати, скільки елементів вона зберігає.

#### Звичайний масив вказівників

Така структура доречна в тому випадку коли є необхідність доступу до елементів за індексом. Найбільш ефективно структура буде працювати у режимі стеку, коли головою стеку є останній елемент масиву.

Вилучення та додавання елементів в середині або на початку масиву призводить до уповільнення роботи структури внаслідок необхідності виконання операцій зсуву.

#### Циклічний масив

Особливість цієї структури в тому, що вона забезпечує швидке додавання та вилучення елементів масиву не тільки у кінці, але й на початку масиву. У такому масиві початковий елемент може знаходитися будь де. Тому структура зберігає значення двох індексів – початок масиву (head) та кінець масиву (tail), рисунок 3.5

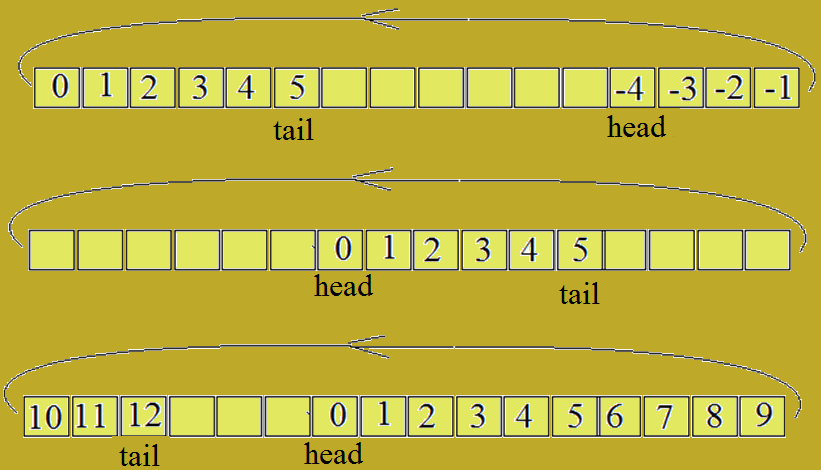


Рисунок 3.5 – Варіанти збереження даних у циклічному масиві

Умовно індекси можуть приймати нібито від’ємні значення. Хоча реально індекс -1 перетворюється у індекс n-1, індекс -2 перетворюється у індексn-2 і т.д. Таке перетворення достатньо просто виконується завдяки представленню від’ємних чисел у додатковому коді.

Така структура найбільш ефективно працює у режимі двосторонньої черги.

#### Бінарна купа

Дані зберігаються в масиві, у вигляді збалансованої бінарної купи: сини елементу черги з індексом n зберігаються в елементах з індексами 2n+1 та 2n+2, рисунок 3.5. Купа не є впорядкованою, але пріоритет кожного вузла вище, ніж пріоритети дітей. Завдяки цьому пріоритет початкового елементу масиву завжди найвищий.

Таку структуру доречно використовувати для реалізації пріоритетної черги.

Вилучення елемента робиться з початку масиву, а на його місце вставляється останній. Після цього, у разі потреби, елементи обмінюються місцями, щоб правило «зверху має бути більший» підтримувалося.

Новий елемент додається в кінець. Після цього також, в разі потреби, елементи міняються місцями.

Кількість обмінів в операції реорганізації масиву невелика і не може бути більше кількості рівнів дерева.



Рисунок 3.5 – Схема бінарної купи

### Комбіновані структури

#### Хеш-таблиці

Хеш-таблиця, всупереч своїй назві, теж є масивом посилань. Але у цьому масиві зберігаються не посилання на дані користувача, а посилання на об’єкт, який називають «кошиком». Кошик містить у собі хеш-код даних користувача, посилання на ці дані і посилання на такий самий кошик, яке спочатку дорівнює null. Таким чином, кошики можуть утворювати лінійний однонаправлений список.

Коли до хеш-таблиці додається новий об’єкт, то спочатку обчислюється унікальний хеш-код цього об’єкту. Наступним кроком є визначення залишку від ділення хеш-коду на довжину масиву посилань на кошики. Цей залишок і буде індексом, за яким має бути розташоване посилання на новий кошик для зберігання посилання на дані користувача. Але внаслідок того, що залишок від ділення хеш-коду не є унікальним, то місце у масиві вже може бути вже зайняте. Таку ситуацію називають колізією. Саме у таких випадках і створюються лінійні списки кошиків, які містять дані користувача, що претендують на той самий елемент масиву.

Для того, щоб зменшити кількість колізій, масив посилань заповнюють частково. Після досягнення критичного розміру створюється новий масив більшого розміру і туди переносяться дані зі старого масиву. Звичайно, що така операція потребує часу.

Хеш-таблиці доцільно використовувати у випадках, коли потрібен швидкий доступ до елементів.

Різновидом хеш-таблиці є асоціативний масив, де дані користувача зберігаються у вигляді пари ключ-значення, і хешування відбувається по ключу.

#### Префіксні дерева

Префіксне дерево фактично є сильно розгалуженим деревом з доступом по ключу. Але значення ключа формується послідовністю усіх батьківських вузлів, кожний з яких зберігає один або кілька елементів ключа. Зазвичай, елементами ключа є символи. Таким чином, нащадки вузла мають загальний префікс, звідки і назва даного абстрактного типу даних. Значення, пов'язані з ключем, не обов’язково пов'язані з кожним вузлом, а тільки з деякими внутрішніми вузлами і з листами.

## Засоби Java для реалізації шару подання

Шар подання для застосування, що створюється, має надавати такі послуги користувачеві програмного додатку:

* відображати дерево складових частин предметної області;
* мати засоби для виклику операцій з деревом, що передбачені технічним завданням;
* надавати користувачеві форми для введення, корегування або перегляду даних, що стосуються вузла дерева;
* відображати результати запитів до дерева у вигляді тексту;
* забезпечити ініціалізацію інтерфейсу після запуску додатку.

Далі коротко розглянуті стандартні засоби та технології Java, що дозволяють вирішувати ці завдання

### Засоби Java для створення та відображення дерева

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

#### 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 3.2.



Рисунок 3.2 – 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.

#### 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.

#### 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.

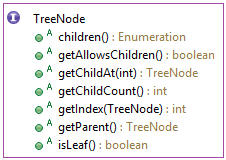


Рисунок 3.3 – Перелік методів інтерфейсу TreeNode

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.

#### Інтерфейс javax.swing.tree.MutableTreeNode

Інтерфейс MutableTreeNode, рисунок 3.4, є розширенням інтерфейсу TreeNode. Методи цього інтерфейсу передбачають можливість працювати з вузлом і з синами вузла використовуючи індекси та посилання.

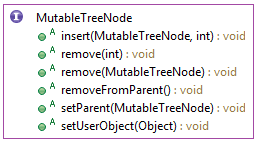


Рисунок 3.4 – Перелік методів інтерфейсу MutableTreeNode

### Засоби для введення/виведення інформації про об’єкти предметної області

Бібліотека Swing надає достатньо багато засобів для введення/виведення інформації. Найбільш популярні компоненти - це JTextField, JTextArea, JList, JCombobox, JCheckBox, JRadiButton. Але якщо застосування передбачає роботу з різними типами об’єктів предметної області, то доцільно компоненти, що стосуються конкретного типу. об’єднати у одну групу. Окрім того, у кожний конкретний момент часу користувач , зазвичай, працює з однією такою групою.

Для створення таких груп існує декілька шляхів.

Достатньо популярним є використання діалогових вікон класу JDialog. У початковому стані діалог має дві кнопки і панель для розташування інформаційних компонентів на якій налаштовано менеджер розташування FlowLayout, рисунок 3.5.

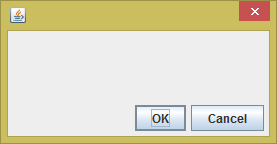


Рисунок 3.5 – Початковий вигляд компоненту JDialog

Появою діалогу на екрані можна керувати за допомогою методу setVisible(boolean). Метод setModal(boolean) дозволяє зробити діалог модальним. У цьому режимі діалог зупиняє виконання коду методу, що викликав метод діалогу setVisible(true). Виконання коду продовжується тільки після закриття діалогу. Це дозволяє дуже просто організувати спілкування користувача з додатком. Мабуть єдиний недолік діалогу полягає у тому, що своєю появою він закриває частину вікна, а інколи це незручно.

Інший спосіб полягає у використанні в межах головного вікна якогось контейнеру для панелей типу Panel. Перевага такого підходу полягає у тому, що панелі не перекривають інформацію головного вікна і не міняють свого положення на екрані.

В якості контейнера можна використовувати, наприклад, компонент JTabbedPane. Метод removeAll() звільнює компонент від панелей, а метод addTab(String, Component) дозволяє розмістити потрібну панелю і навіть вивести текстовий заголовок.

Як контейнер можна використовувати і будь яку панель, налаштувавши у неї менеджер розташування CardLayout і після цього розташувати там усі потрібні панелі. Невелика незручність цього методу в тому, що цей менеджер розташування потребує, щоб кожній панелі було присвоєно унікальне ім’я в момент передачі панелі менеджеру. Це ім’я дозволяє активізувати панелі у довільному порядку.

Головний недолік використання звичайних панелей полягає у тому, що програмісту доведеться самому реалізовувати зупинку виконання методу, що активізував інформаційну панель. За звичай, ця задача вирішується використанням потоків і засобів їх зупинки та відновлення їх роботи.

### Засоби Java для виклику операцій користувача над деревом

Найчастіше для виклику опреацій користувача у графічних інтерфейсах використовуються кнопки та меню.

#### Кнопки

Клас JButton реалізує звичайну кнопку. Напис на кнопці налаштовується методом setText(String) або через вікно properties. Подія натискання на клавішу має назву actionPerformed.

#### Радіоклавіши

Клас JRadioButton дозволяє створити групу радіоклавіш для реалізації багатоваріантного вибору.

При роботі з радіоклавішами слід враховувати, що для об'єднання радіоклавіш в одну групу необхідно використовувати об'єкт типу GroupButton.

  Найпростіше створити групу радіоклавіш за допомогою візуального редактора. Для цього слід виділити групу об'єднуються радіоклавіш використовуючи комбінацію клавіші Ctrl і MouseClickс, а потім в контекстному меню вибрати функцію setButtonGroup. Цей об'єкт можна прописати і вручну, використовуючи конструктор цього класу, і потім кожну радіоклавішу після створення додати в цю групу, використовуючи метод add.

Перевірити стан кнопки можна за допомогою методу isSelected(). Вибір кнопки фіксується подією actionPerformed.

#### Кнопки меню

Кнопки меню є об’єктами класу JMenuItem, що створений на основі звичайної кнопки типу JButton. Відповідно до цього і обробка подій для кнопок меню така ж сама, як і для звичайних кнопок.

Але кнопка меню не використовується сама по собі. Вона є складовою частиною об’єктів меню. Для реалізації меню можна використовувати декілька класів.

Клас JMenuBar використовується для створення головного меню додатку. Об’єкт цього класу стандартно розташовується у верхній частині вікна, займаючи весь рядок. До цього об’єкту стандартно додаються об’єкти типу JMenu, які можуть використовуватися як кнопки, але частіше використовується для розміщення груп елементів типу JMenuItem.

Клас JPopupMenu використовується для створення контекстного меню. Контекстне меню можна пов’язати з різними компонентами. При цьому e до класу додатку автоматично додається статичний метод addPopup. Після створення контекстного меню можна додати до нього потрібну кількість об’єктів типу JMenuItem – кнопок меню. Для зручності користування до меню можна додавати роздільники – об’єкти типу JSeparator.

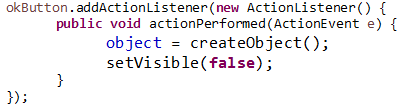
Окрім кнопок JMenuItem у меню можна використовувати кнопки типу JRadioButtonMenuItem, які є аналогами радіоклавіш.

#### Обробка події натискання на кнопку

Коли користувач додатку натискає на кнопку, кнопка формує подію actionPerformed. Але для того, щоб у відповідь на натискання кнопки виконувався деякий код, необхідно створити слухача (listener) цієї події і зареєструвати його у кнопки.

Для реєстрації слухача використовується метод кнопки addActionListener, а слухач має реалізовувати інтерфейс ActionListener методом actionPrformed.

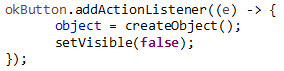
Процес створення такого об’єкту, його реєстрація та реалізація коду реакції на подію зазвичай об’єднують у одну конструкцію з використанням анонімних об’єктів анонімних класів, створених на основі інтерфейсів. Наприклад:



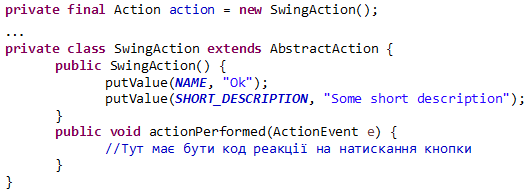
Розглянута конструкція створюється за допомогою Eclipse автоматично. Програмісту залишається дописати тільки тіло методу.

У цьому прикладі до кнопки okButton за допомогою методу addActionListener() додається новий слухач типу ActionListener. Цей слухач у відповідь на подію натискання на кнопку буде виконувати метод actionPerformed.

В Java8 розглянута конструкція дещо спрощується за рахунок використання лямбда функції



Ще один варіант реалізації реакції на події кнопок – створення об’єктів типу AbstractAction. Для створення такого об’єкту слід вибрати функцію setAction -> new з контекстного меню кнопки. Далі з’являється меню, що пропонує або вибрати раніше створений Action, або створити новий. У останньому випадку результатом буде створення внутрішнього класу і поля, що містить посилання на об’єкт цього класу.



Програмісту залишається тільки написати код реакції на подію, реалізувавши тіло методу actionPerformed. Окрім того, можна змінити назви класу, змінної, назву кнопки та короткий опис реакції.

Головна перевага цього способу полягає у тому, що той самий «action» можна пов’язати з декількома кнопками, елементами меню, тощо.

Можливий і спрощений варіант створення об’єкту Аction, але він реалізується вручну. Після створення такого об’єкту він буде доступний у меню setAction кнопки і його можна буде призначати для будь якої кількості кнопок і елементів меню.

**private** AbstractAction actAdd = **new** AbstractAction("Add node") {

@Override

**public** **void** actionPerformed(ActionEvent e) {

onAdd();

}

};

Окрім того, у наведеному вище безіменному класі можна визначити блок ініціалізації, де встановити значення для констант об’єкту Action за допомогою методу putValue.

# ПРИКЛАД ПОБУДОВИ СИСТЕМИ

У наведеному прикладі створюється Java проект, який забезпечує роботу з інформацією про університет, яку представлено у вигляді дерева.

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

Рівень 1 - «факультет», зберігає інформацію про назву факультету та ім’я декана. Дані про факультети мають зберігатися у колекції TreeSet.

Рівень 2 - «кафедра», містить назву кафедри, ім’я завідувача, та ознаку - чи є кафедра випускаючою. Перелік кафедр має зберігатися у масиві.

Рівень 3 «викладач», містить ім’я викладача, посаду, дату народження. Інформація про викладачів має зберігатися у однонаправленому списку.

Дані в усіх структурах даних мають бути впорядковані.

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.

У проекті має бути реалізований інтерфейс користувача, що надасть можливість користувачеві додатку виконувати усі заявлені операції

## Аналіз задачі та проектування системи

Реалізація проекту вимагає вирішення декількох задач. Розглянемо ці завдання по черзі.

### Модель предметної області

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 спільну для всіх перелічених абстракцій поведінку. Наявність такого типу дозволить створювати узагальнені методи для обробки інформації про предметну область.

На діаграмі класів, рисунок 4.1, узагальненим типом є інтерфейс IUserData, а класами, що відтворюють абстракції «Університет», «Факультет», «Кафедра» та «Викладач», є класи UniverData, FacultyData, DeptData, TeacherData.

Для забезпечення можливості серіалізації об’єктів інтерфейс IUserData успадковує інтерфейс Serialisable.

Для підтримки операцій впорядкування даних реалізуймо також інтерфейс Comparable.

Приватні параметри об’єктів будемо задавати через конструктори, а для публічного доступу до параметрів створімо тільки геттери.

У кожному класі сформуємо також метод ToString().

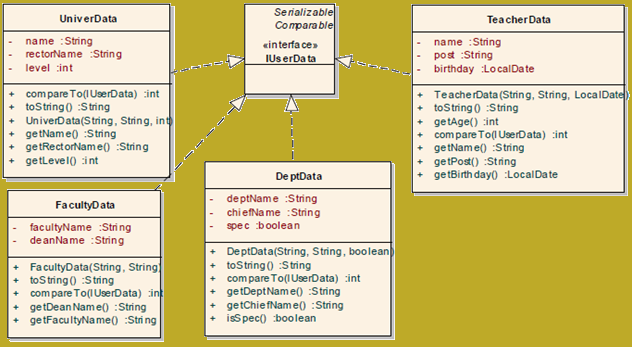


Рисунок 4.1 – Діаграма класів предметної області

### Інтерфейс користувача

Візуальну частину будемо створювати відповідно до вимог технічного завдання. Дерево будемо відображати за допомогою об’єкту класу JTree.

Для виклику потрібних операцій створімо головне меню на основі класу JMenuBar та контекстне меню дерева, що буде дублювати головне меню.

Для відображення інформації про об’єкти предметної області та для виведення результатів запитів будемо використовувати панелі компоненту JTabbedPane. Очікуваний вигляд інтерфейсу наведено на рисунку 4.2.

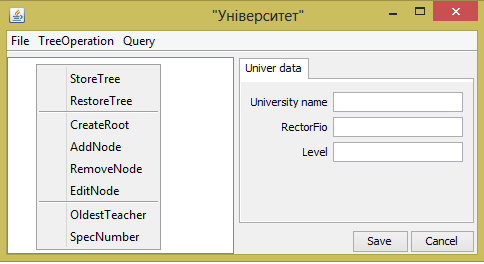


Рисунок 4.2 – Інтерфейс користувача

На цьому рисунку показано головне меню та контекстне меню дерева. Групи контекстного меню відповідають складовим головного меню.

Праворуч показано панель для введення/виведення інформації про університет. Аналогічні панелі мають бути створені і для інших об’єктів предметної області. Окрім того, має бути створено панель для виведення запитів з компонентом TextArea та панель із заставкою проекту, що має відображатися, коли жоден з вузлів дерева не вибрано.

На рисунку 4.3 показано діаграму класів, що використовуються для реалізації графічного інтерфейсу користувача.

Панелі для введення та відображення інформації, що містять об’єкти предметної області, узагальнені інтерфейсом IDataPanel.

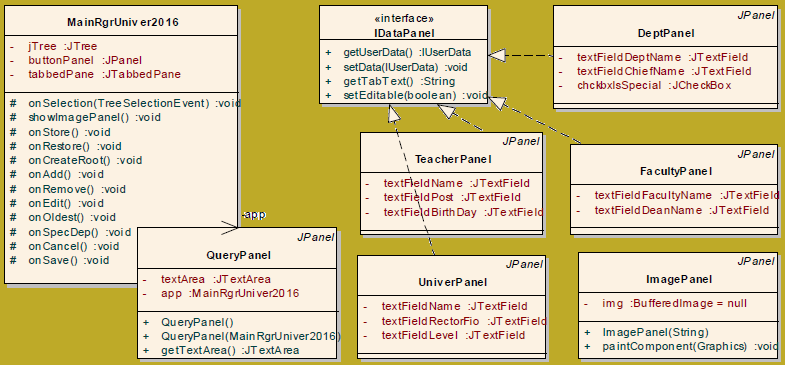


Рисунок 4.3 – Діаграма класів шару подання

### Модель дерева

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 4.4.

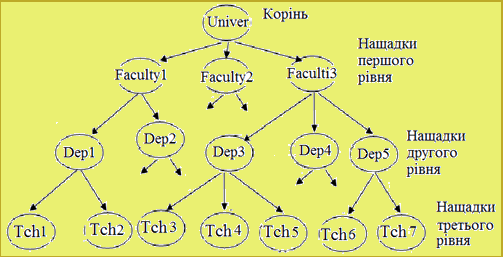


Рисунок 4.4 – Логічна структура інформаційної системи

Відповідно до завдання, нащадки різних рівнів мають зберігатися у різних структурах даних. Таку логічну структура не можна реалізувати за допомогою стандартних класів Java для представлення моделі дерева, наприклад, об’єкту класу DefaultMutableTreeNode. Однак можна використовувати стандартний інтерфейс для вузлів дерева TreeNode, та стандартний інтерфейс для моделі дерева TreeModel. 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 для вузлів дерева, які будуть містити абстракції «Університет», «Факультет», «Кафедра» та «Викладач». Цє класи UniverNode, FacultyNode, DeptNode, TeacherNode. Ці класи мають реалізувати інтерфейси Serializable, Iterable, Comparable та TreeNode. Окрім того доцільно створити для цих вузлів інтерфейс, що буде визначати вимоги нашого проекту до вузлів. Зокрема треба передбачити можливість створення діалогових панелей для введення або відображення інформації про себе і про сина. Це буде інтерфейс INode. Доцільно також створити узагальнений клас, у якому реалізувати узагальнені методи інтерфейсів, якщо є така можливість. Це клас AniNode.

Спільним атрибутом для перелічених абстракцій може бути об’єкт типу IUserData, а однакова поведінка полягає у

На діаграмі класів, рисунок 4.5, показані ці інтерфейси та класи. Методи класів останнього рівня ієрархії показані тільки для класу UniverNode. У решти класів цього рівня методи ті ж самі, реалізації, звичайно, різні.

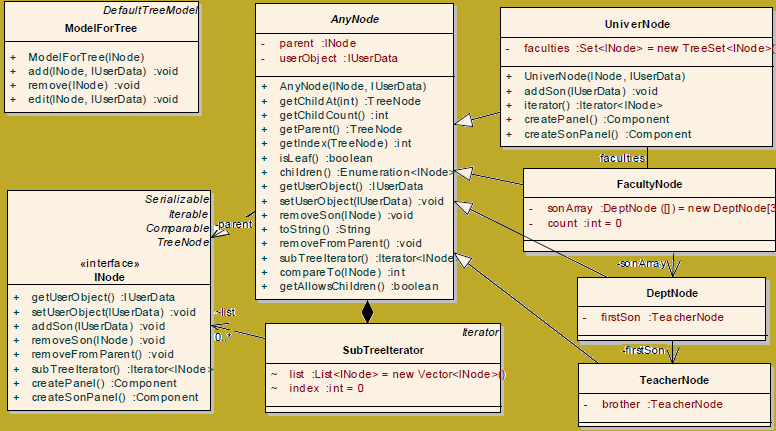


Рисунок 4.5 – Діаграма класів моделі дерева

Модель дерева представлена класом MForTree.

## Структура пакетів проекту

Приклад РГР реалізовано у вигляді проекту OopRgr2017.

Класи та ресурси проекту розташовані у чотирьох пакетах:

* dataModel, для класів та інтерфейсів, що реалізують абстракції предметної області – університет, факультет, кафедра, викладач;
* treeModel, для класів та інтерфейсів, що визначають структуру дерева, що використовується для відображення предметної області;
* view, для класів та інтерфейсів, що реалізують візуальну частину проекту;
* resource, для додаткових ресурсів, зокрема для файлу .jpg заставки. Ресурси проекту, так само як і класи, слід зберігати у пакетах.

## Реалізація класів для об’єктів предметної області

Реалізацію проекту почнемо із створення класів та інтерфейсів, що стосуються абстракцій предметної області.

Об’єкти саме цих класів будуть містяться у вузлах дерева. Ці об’єкти є реалізаціями абстракцій «університет», «факультет», «кафедра» та «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, класи повинні мати загальний інтерфейс, методи якого визначать узагальнені дії для цих об’єктів.

Інтерфейс та класи, що будуть створюватися розташуємо в пакеті dataModel.

### Інтерфейс IUserData

Створімо засобами Eclipse інтерфейс IUserData, що буде реалізовано у класах, що відтворюватимуть «університет», «факультет», «кафедру» та «викладача». Головна цінність цього інтерфейсу у тому, що він може використовуватися як тип для об’єктів усіх класів предметної області. А це дасть змогу писати поліморфний код.

Успадкуємо у нашому інтерфейсі також інтерфейси Serializable та Comparable. Перший потрібен для того, щоб мати можливість серіалізувати дерево разом з даними вузлів.

Другий може знадобитися, якщо доведеться порівнювати дані, наприклад, у разі необхідності впорядкувати вузли.



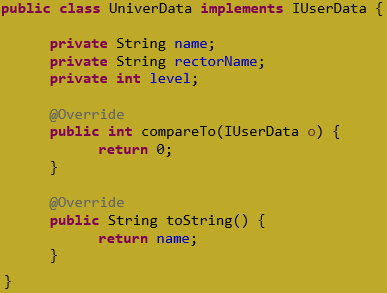
### Клас UniverData

Створімо клас UniverData, що відтворює абстракцію «університет» і реалізує інтерфейс IUserData.

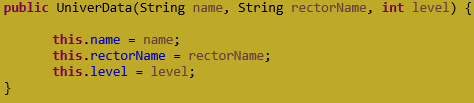
У класі визначимо три поля – для назви університету, для імені ректора, та рівня університету.

Визначимо також метод toString(), який буде повертати ім’я університету. Саме цей метод буде використовуватися для відображення університету у вузлу дерева.

Об’єкт класу UniverData буде зберігатися у корені дерева. А корінь присутній у дереві в єдиному екземплярі, і його порівнювати нема з чим. Тому визначимо, що метод compareTo, реалізації якого вимагає Comparable, завжди буде повертати нуль.



Далі, за допомогою засобів Eclipse створімо у класі конструктор з трьома параметрами, що відповідають полям класу:

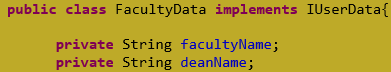


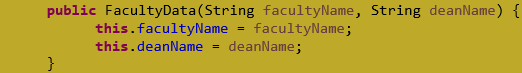
Згенеруймо також геттери для усіх полів.

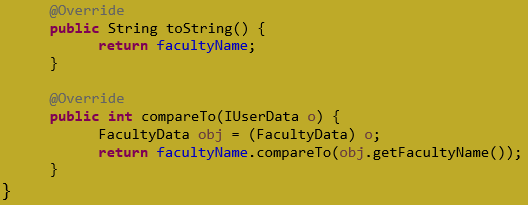
### Клас FacultyData

Аналогічно попередньому сформуємо клас FacultyData, що відтворює абстракцію «факультет» і реалізує інтерфейс IUserData.

У класі визначимо два поля – для назви факультету та для імені декана. Далі згенеруємо конструктор з полями, геттери та метод toString() по назві факультету, а також метод порівняння. Нижче показано частину класу:





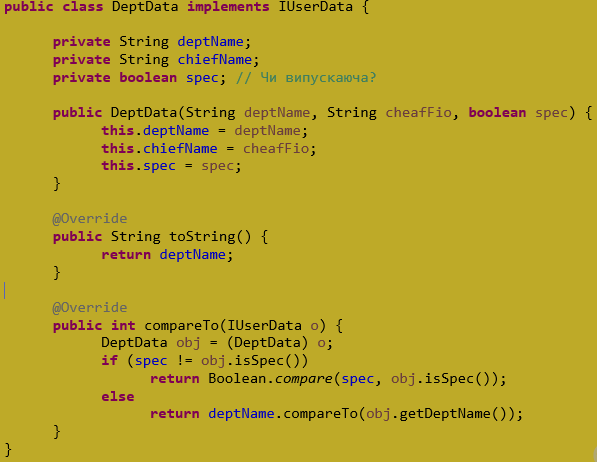


### Клас DeptData

Аналогічно до попередніх класів створімо клас DeptData, що відтворює абстракцію «кафедра» і реалізує інтерфейс IUserData.

У класі визначимо поля, що визначають назву кафедри, ім’я завідувача, та ознаку, чи є кафедра випускаючою.

Нижче наведено частину тексту класу. Геттери та заготовки методів інтерфейсу IUserData не показано.

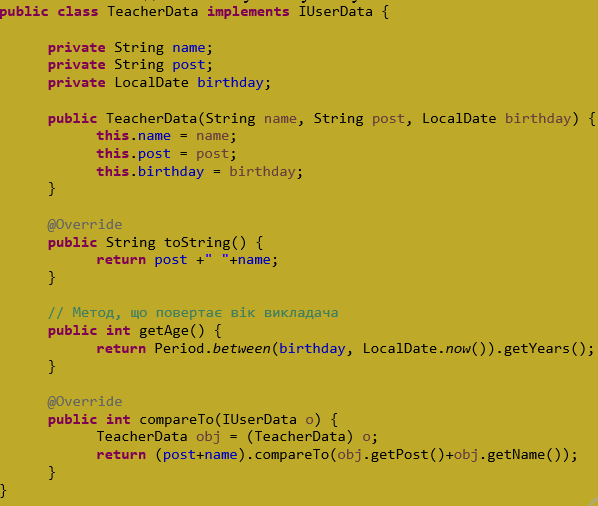


### Клас TeacherData

Аналогічно до попередніх класів створімо клас TeacherData, що відтворює абстракцію «викладач» і реалізує інтерфейс IUserData.

У класі визначимо поля, що визначають ім’я викладача, його посаду та дату народження.

Нижче наведено частину тексту класу.



У наведеному тексті не показано геттери та методи інтерфейсу IUserData, натомість наведено метод getAge(), що повертає вік викладача.

## Дизайн головного вікна

Реалізацію проекту продовжимо розробкою дизайну візуальної частини. По-перше це сформує чіткий перелік завдань першого рівня, по-друге ми отримаємо можливість виконувати часткове тестування проекту.

Для класів, що пов’язані з реалізацією інтерфейсу користувача будемо використовувати пакет view. Клас можна назвати MainRgrUniver.

На рисунку 4.6 зображено можливий початковий вигляд візуальної частини додатку.

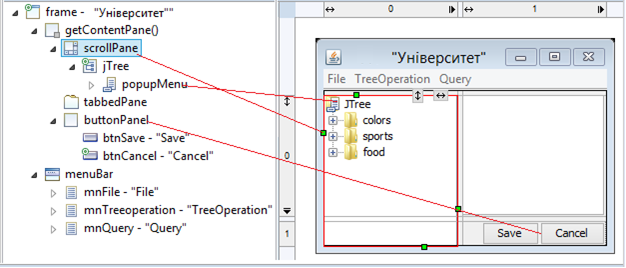


Рисунок 4.6 – Візуальна частина на першому етапі розробки

У верхній частині головної панелі вікна встановлено компонент JMenuBar і до нього додано три елементи типу JMenu (File, TreeOperation, Query). Ми не будемо поки що формувати кнопки, що будуть визначати перелік функцій кожного з цих меню. Зробимо пізніше це програмним шляхом.

Для головної панелі налаштовано менеджер компоновки GridBagLayout з двома колонками і двома рядками. Рядок 0 та колонки мають налаштування Grow, що надає можливість змінювати розміри панелей у разі зміни розмірів вікна.

Рядки 0 та 1 колонки з номером 0 займає JScrollPane на якій встановлено компонент класу JTree. На компонент JTree встановлено меню типу JPooupMenu, перелік функцій якого теж пізніше сформуємо програмним шляхом.

Рядок 0 колонки з номером 1 займає панель типу JTabbedPane. Закладки для цієї панелі ми також будемо створювати і додавати програмним шляхом.

Рядок 1 колонки 1 займає панель JPanel з менеджером компоновки FlowLayout. На панелі встановлено дві кнопки.

На завершення першого етапу роботи з візуальною частиною треба за допомогою функції Expose Component контекстного меню компонентів зробити доступними компоненти JTree, JTabbedPane та JPanel і для зручності роботи поміняти назви деяких компонентів.

Для зміни назв компонентів можна використовувати функцію Rename контекстного меню компонента.

Компонент JTree для відображення дерева назвемо jTree для того, щоб ім’я об’єкту відповідало назві класу. Панель для кнопок назвемо buttonPanel. Кнопкам дамо назви btnSave та btnCancel.

### Використання об’єктів Action для меню та кнопок

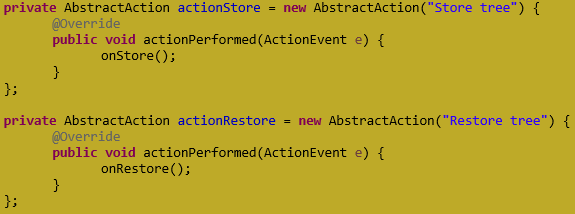
#### Створення об’єктів типу Action

У нашому додатку функції, що пов’язані з обробкою дерева, можна буде викликати або через головне меню, або через контекстне меню дерева. У таких випадках доцільно створювати відповідні об’єкти Action і потім пов’язувати їх з компонентами JMenuItem. Про це йшлося у підрозділі «Засоби Java для реалізації шару подання».

Перейдемо до тексту класу і продовжимо перелік полів визначеннями об’єктів типу Action, скориставшись спрощеним способом для створення таких об’єктів з використанням анонімного класу, що розширює клас AbstractAction.

Нижче наведено опис тільки двох таких полів – для кнопок меню «Зберегти дерево у файлі» та «Відновити дерево з файлу», що будуть пов’язані з меню File, а також будуть входити до складу контекстного меню дерева.

Створюючи ці об’єкти, використовуйте засоби автодоповнення коду.



Зверніть увагу на те, що об’єкти використовують звернення до методів візуальної частини onStore() та onRestore(), яких поки що нема. Відповідно, Eclipse сповіщає про ці помилки. Тож слід скористатися засобами автоматичного виправлення помилок і згенерувати protected заготовки для цих методів. Не варто використовувати функцію створення private методу. Працювати з такими методами буде незручно.

Реалізовувати ці методи будемо пізніше.

Окрім двох об’єктів Action, що наведені вище, слід створити решту таких об’єктів. У нашому проекті це будуть об’єкти для наступних дій:

* створення кореня дерева (actionCreateRoot);
* додавання вузла дерева (actoinAdd);
* вилучення вузла древа (actoinRemove);
* редагування даних вузла дерева(actoinEdit);
* запит на пошук найстарішого викладача (actoinOldest);
* підрахунок кількості спецкафедр для факультетів (actoinSpecDep).
* реакція на кнопку Save (actionSave)
* реакція на кнопку Cancel (actionCancel)

#### Зв’язування об’єктів типу Action з кнопками

Кнопки «Save» та «Cancel» можна пов’язати з об’єктами Actoin у режимі дизайну. Для цього слід скористатися функцією setAction контекстного меню кнопки і вибрати потрібний Action.

#### Зв’язування об’єктів типу Action з меню

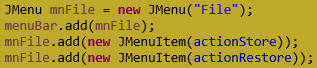
Для кнопок меню, яких ще нема, краще скористатися програмним способом. У цьому разі коду буде менше, а це полегшує роботу з текстом класу.

Знайдіть у тілі класу рядки, де створюється об’єкт типу JMenu для функції File і додається до головного меню. Краще за все це зробити клацнувши по відповідному компоненту меню у режимі дизайну, а потім перейти у режим роботи з текстом класу.

Ці рядки можуть виглядати так:

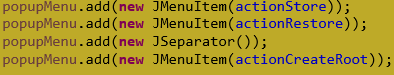


Після цих рядків слід дописати код, де створюються нові об’єкти JMenuItem з використанням відповідних об’єктів Action та додаються до меню. Після доповнення текст може виглядати так:



Аналогічно слід додати необхідні функції і до інших складових головного меню.

Так само можна додати функції і для контекстного меню дерева. Нижче наведено фрагмент такого коду:



Передостанній рядок коду додає до меню роздільник між групами функцій.

Після завершення цієї роботи можна активізувати додаток і перевірити, чи відкриваються потрібні функції меню. Але для того, щоб щось відбулося після натискання на кнопку меню, слід додати якийсь код до заготовок методів, що були створені для об’єктів Action, наприклад, до методу onCreateRoot().

## Створення класів панелей для компоненту JTabbedPane

Питання вибору засобів для обміну інформацією між користувачем додатку і програмою розглядалося у підрозділі «Засоби Java для реалізації шару подання». У нашому додатку ми будемо використовувати панель типу JTabbedPanel як контейнер для розміщення панелей, через які користувач буде вводити інформацію, або отримувати її.

У додатку потрібні будуть такі панелі:

* панель заставка, що буде відображатися на початку роботи додатку, та у випадку, коли не вибрано жодного вузла дерева;
* панель для введення/виведення інформації про університет;
* панель для введення/виведення інформації про факультет;
* панель для введення/виведення інформації про кафедру;
* панель для введення/виведення інформації про викладача;
* панель для виведення інформації про результати запитів.

Усі перелічені вище панелі будемо створювати як об’єкти класів, що успадковують клас JPanel.

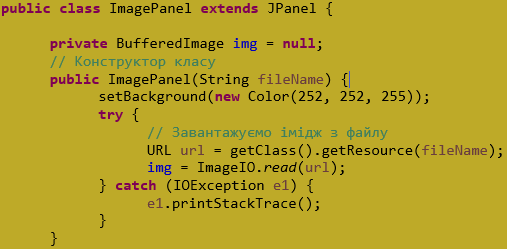
### Панель заставки

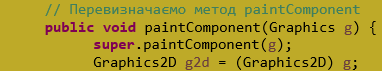
#### Клас ImagePanel

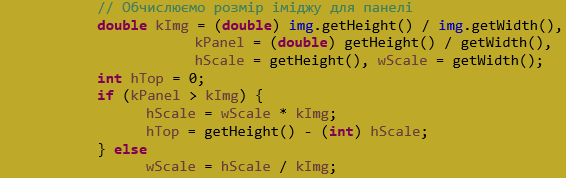
Панель-заставки може мати будь який вигляд. У нашому додатку ми відобразимо на ній фото університету, яке буде масштабуватися відповідно до розмірів вікна. На закладці панелі виведемо текст «This is my university».

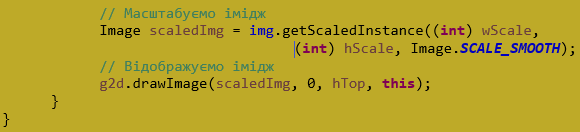
Текст класу ImagePanel для панелі заставки наведено нижче. Клас розташовуємо у пакеті view, де знаходяться класи, що пов’язані з візуальною частиною проекту.

Клас можна написати і простіше, якщо заповнювати вікно, не зважаючи на деформацію зображення.









#### Тестування класу ImagePanel

Перш за все треба знайти файл зображення типу .jpg або .png, що має відображатися на панелі. Хай це буде файл stu.jpg.

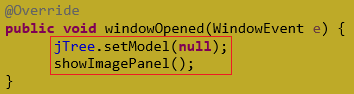
Далі слід імпортувати у пакет resource вибраний файл.

Після цього створімо у класі візуальної частини поле imagePanel і одразу проініціалізуємо його:



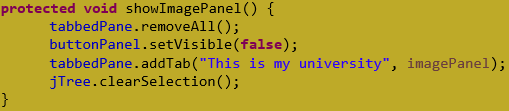
Для того, щоб панель-заставка відображалася після запуску нашого додатку, потрібно через функцію контекстного меню вікна «Add event handler» для вікна додатку створити слухача події windowOpened.

Після цього доповнити метод windowOpened у створеному анонімному класі обробника кодом, що наведено далі (виділено рамочкою).



Перший рядок коду ліквідує модель дерева, що використовувалася за замовченням. Тому компонент JTree на початку нічого не буде відображувати.

А у другому рядку викликається метод, якого ще нема. Тому шляхом автоматичного виправлення помилки слід згенерувати заготовку цього **protected** методу і додати до методу код, що забезпечує відображення панелі:



Другий рядок у тілі методу робить невидимою панель з кнопками тому що у цьому режимі кнопки не потрібні.

Після цього можна активізувати додаток. Після відкриття вікна панель має відобразитися у контейнері JTabbedPane, рисунок 4.7.

На рисунку також показано контекстне меню дерева, що з’явилося після натискання на праву кнопку миші.

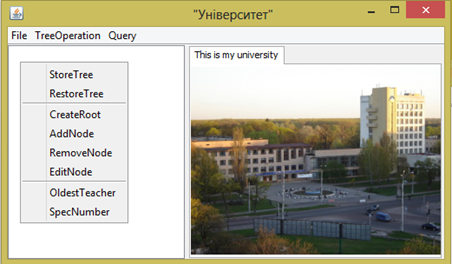


Рисунок 4.7 – GUI додатку після старту з контекстним меню дерева

### Панель для виведення результатів запитів

На панелі потрібно мати компонент JTextArea для виведення результатів запитів у вигляді тексту та кнопку за закриття панелі.

#### Клас QueryPanel

Створімо візуальний клас QueryPanel на основі класу JPanel і розташуємо на ньому потрібні компоненти так, як це показано на рисунку 4.8.

Для створення візуальної композиції використовується менеджер компоновки GridBagLayout.

Для забезпечення доступу об’єктам інших класів до області виведення тексту слід скористатися функцією контекстного меню Expose component. Внаслідок цього буде сформовано метод getTextArea(), що реалізує такий доступ.

Після цього треба перейти до режиму корегування тексту класу.

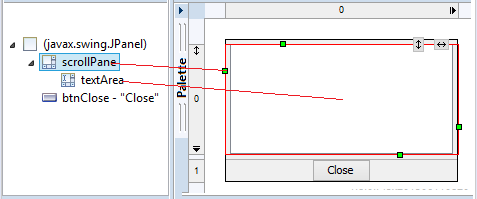


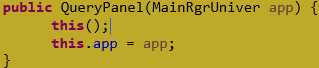
Рисунок 4.8 – Панель QueryPanel у режимі дизайну

У тілі класу створімо поле app типу MainRgrUniver. Тут буде зберігатися посилання на візуальної частини проекту.

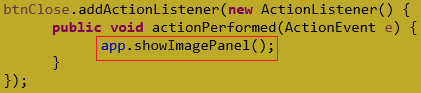
Це поле забезпечить доступ панелі до методів головного вікна.



Далі згенеруємо конструктор класу, використовуючи поле app і додаймо до згенерованого тексту виклик конструктора без параметрів **this()** цього ж класу.



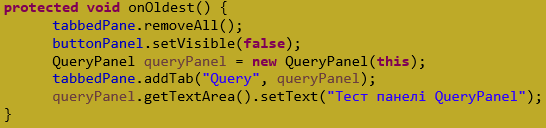
Після цього можна запрограмувати реакцію на кнопку Close.



Рамочкою виділено текст, який потрібно додати до згенерованого обробника події actionPerformed.

#### Тестування класу QueryPanel

Додаймо до методу onOldest() код для тестування панелі:



Тепер можна активізувати додаток. Після виклику функції меню Query→OldestTeacher має з’явитися панель QueryPanel з текстом «Тест панелі QueryPanel».

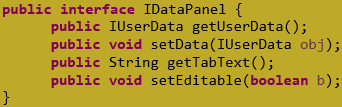
Після натискання на кнопку «Close» панель має зникнути і з’явитися панель-заставка.

### Панелі введення/виведення

Функціонально, панелі для введення/виведення інформації про університет, факультет, кафедру та викладача мають багато спільного. Тому доцільно цей функціонал виділити у спільний для цих панелей програмний інтерфейс. Це дозволить писати узагальнений код для роботи з панелями.

#### Інтерфейс IDataPanel для панелей введення/виведення

То ж створімо інтерфейс IDataPanel у пакеті view.



Перший та другий методи цього інтерфейсу використовують раніше створений тип IUserData, який охоплює усі типи даних предметної області.

Метод getUserData() має повертати об’єкт, що створено з використанням даних користувача. Вибір конструктора для створення об’єкту залежить від панелі.

Метод setData(IUserData) має відобразити на панелі дані про об’єкт, що передано як параметр.

Метод getTabText() має повертати текст, з яким буде відображатися панель, як закладка tabbedPane.

Метод setEditable(boolean) керує можливістю вводити інформацію до текстових полів панелі.

#### Клас UniverPanel для введення даних про університет

Це візуальний клас, що вирішує завдання введення/виведення даних про університет, рисунок 4.9.

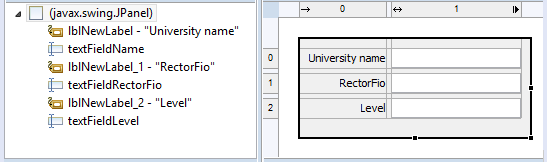


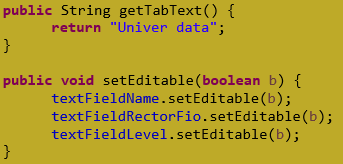
Рисунок 4.9 – Панель UniverPanel у режимі дизайну

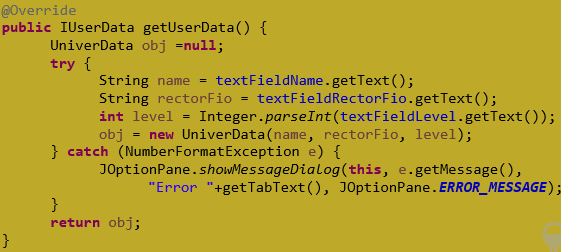
Клас створено на основі класу JPanel. Для зручного розташування елементів панелі доцільно вибрати менеджер розташування GridBagLayout.

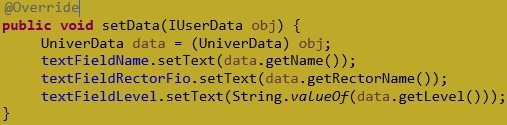
Призначення компонентів JTextField зрозуміло з тексту компонентів JLabel.

Текстові поля слід зробити доступними за допомогою функції контекстного меню Expose component.

Після створення візуальної композиції почнемо роботу з кодом класу. Заявимо у класі реалізацію інтерфейсу IDataPanel та реалізуємо його методи:



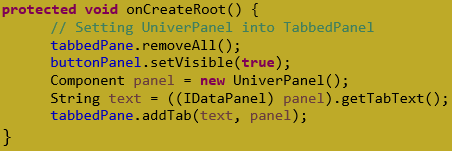




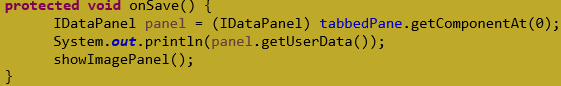
#### Тестування класу UniverData

Для тестування цієї панелі скористаймося кнопкою меню CreateRoot. Метод onCreate, який пов’язаний з цією кнопкою реально буде складнішим, ніж ми зараз його реалізуємо. Та для тестування панелі такої реалізації достатньо.

Нижче наведено цю спрощену реалізацію, яка забезпечить появу панелі університету на екрані:



Далі додаймо такий код до методу onSave. Це теж його тимчасова реалізація:



Слід також тимчасово запрограмувати реакцію на кнопку «Cancel».



Тепер можна активізувати додаток. Після виклику функції меню «CreateRoot» має з’явитися панель університету і кнопки «Save» та «Cancel».

Якщо до панелі нічого не вводити і натиснути кнопку «Save», то отримаємо віконце з повідомленням про помилку, а після його закриття на консолі має з’явитися «null».

Якщо ввести допустимі дані до полів панелі, то на консолі має з’явитися результат виклику методу toString для сформованого об’єкту а у tabbedPane з’явится панель-заставка.

Якщо натиснути «Cancel» то нічого не буде виводитися і з’явиться панель-заставка.

#### Клас FacultyPanel для роботи з даними про факультет

Панель створюємо за аналогією з попередньою. Зовнішній вигляд панелі показано на рисунку 4.10.

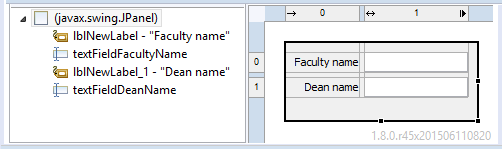


Рисунок 4.10 – Панель FacultyPanel у режимі дизайну

Реалізація інтерфейсу IDataPanel дуже схожа на реалізацію цього інтерфейсу у класі UniverPanel. Тому ці методи тут не наводимо.

#### Тестування панелі FacultyPanel

Для тестування цієї панелі можна скористатися методом onCreateRoot, який було створено для тестування панелі UniverPanel. У цьому методі потрібно тільки внести зміни до рядка з викликом конструктора, замінивши UniverPanel на FacultyPanel.

#### Клас DeptPanel для роботи з даними про кафедру

Зовнішній вигляд панелі показано на рисунку 4.11.

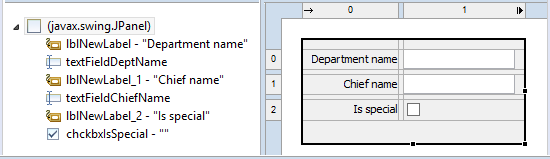


Рисунок 4.11 – Панель DeptPanel у режимі дизайну

Реалізація інтерфейсу IDataPanel дуже схожа на реалізацію цього інтерфейсу у попередніх класах. Особливість полягає у тому, що компонет JCheckBox не підтримує метод setEditable(boolean), натомість треба використовувати метод setEnabled(boolean). Визначити стан цього елементу можна за допомогою методу isSelected(), а змінити стан можна за допомогою методу setSelected(boolean).

#### Тестування панелі DeptPanel

Для тестування цієї панелі теж можна скористатися методом onCreateRoot. Слід тільки замінити ім’я конструктора на DeptPanel.

#### Клас DeptPanel для роботи з даними про кафедру

Зовнішній вигляд панелі показано на рисунку 4.12. Відмінність цієї панелі від попередніх у тому, що для введення дати народження використовується текстове поле з налаштованою властивістю TitledBorder.

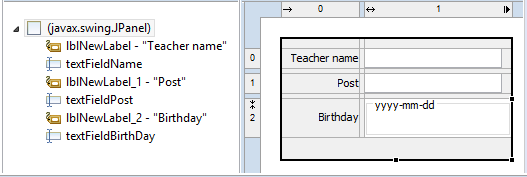
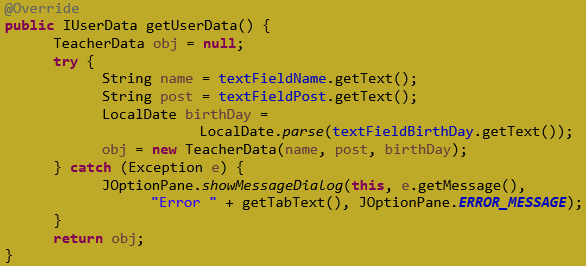
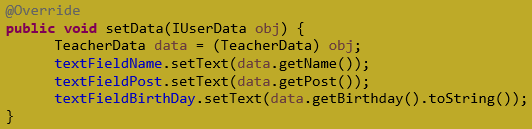


Рисунок 4.12 – Панель TeacherPanel у режимі дизайну

Особливості реалізації інтерфейсу IDataPanel для цього класу полягають у роботі з типом LocalData. Тому наводимо реалізації методів цього інтерфейсу у яких використовується дата:





#### Тестування панелі TeacherPanel

Для тестування цієї панелі теж можна скористатися методом onCreateRoot. Слід тільки замінити ім’я конструктора на TeacherPanel.

Після тестування доцільно змінити ім’я конструктора на UniverPanel, бо саме ця панель потрібна для створення кореня дерева.

## Реалізація класів для вузлів дерева

В основі моделі дерева лежать класи його вузлів. Об’єкти цих класів використовуються як контейнери до даних користувача дерева. Відповідно до технічного завдання класи для різних рівнів дерева мають бути побудовані з використанням різних структур даних. Але незалежно від цього усі вузли дерева, незалежно від рівня, мають реалізовувати однаковий інтерфейс.

Тож визначимося з цим інтерфейсом. Інтерфейс і класи будемо розташовувати у пакеті treeModel.

### Інтерфейс для вузлів дерева

Інтерфейс визначає загальні вимоги до вузлів.

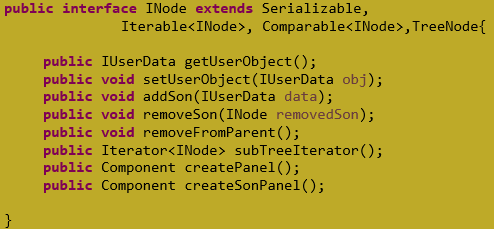
Вузли разом з деревом потрібно буде зберігати у файлі, тому наш інтерфейс має успадковувати Serialisable.

Вузли треба порівнювати, тому успадковуємо Comparable.

Кожний вузол має надавати стандартний доступ до списку своїх синів, тому успадковуємо Iterable.

Вузол має реалізовувати інтерфейс TreeNode, щоб його можна було передавати у конструктор компоненту JTree як корінь дерева для відображення.

До вимог, що закладені у стандартні інтерфейси, додаймо ще й свої:



Перші два методи надають доступ до даних, що зберігаються у дереві.

Методи addSon(IUserData) забезпечує створення і додавання сина із заданими даними користувача до вузла.

Метод remove(INode) забезпечує видалення заданого сина даного вузла.

Метод removeFromParent() дозволяє вилучити сам вузол.

Метод subTreeIterator() повертає об’єкт типу Iterator, що забезпечує перегляд усіх нащадків вузла, включаючи його самого.

Метод createPanel створює панель введення/виведення даних користувача для даного вузла.

Метод createSonPanel створює панель введення/виведення даних користувача для сина вибраого вузла. Ця панель потрібна для реалізації иетоду додавання сина.

### Узагальнений клас для вузлів дерева

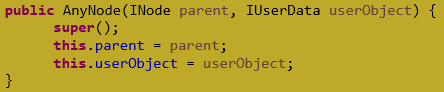
Цей клас буде батьківським для усіх типів вузлів нашого дерева. До нього ми винесемо загальні поля і реалізуємо, по можливості, деякі методи.

Створімо клас AnyNode, що реалізує інтерфейс INode і шляхом автоматичного виправлення помилок реалізуємо у ньому заготовки для методів усіх успадкованих інтерфейсів.

Далі визначимо поля, що необхідні вузлам незалежно від їх реалізації. Таких полів два. Вони зберігають посилання на батьківський вузол та на дані користувача:



Згенеруємо конструктор з цими полями:

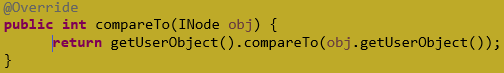


Тепер, по черзі, розглянемо методи інтерфейсів, що підлягають реалізації, і якщо можна, то реалізуємо їх. Для решти перенесемо реалізацію до класів спадкоємців.

Інтерфейс Serializable не потребує реалізації, тому почнемо з інтерфейсу Comparable.

#### Реалізація інтерфейсу Comparable

Єдиний метод цього інтерфейсу можна реалізувати, використовуючи методи порівняння для даних користувача:



#### Реалізація інтерфейсу Iterable

Інтерфейс Iterable потребує реалізації методу iterator, що повертає об’єкт типу Iterator. Але реалізація цього методу залежить від способу збереження нащадків вузлів і цю проблему не можна вирішити однаково для усіх вузлів. Тому метод iterator() ми вилучимо з тіла класу, а клас зробимо абстрактним. Та це не означає, що ми не зможемо користуватися методом iterator() у класі AnyNode. Адже цей метод заявлено в інтерфейсі.

#### Реалізація інтерфейсу TreeNode

Перелік методів цього інтерфейсу для зручності наведено на рисунку 4.13

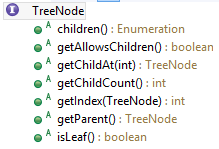
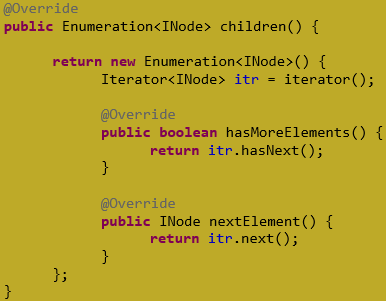
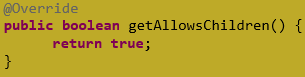


Рисунок 4.13 – Методи інтерфейсу TreeNode

Метод children(), що повертає ітератор Enumeration можна реалізувати через ітератор Iterator.



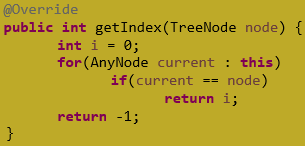
Метод getAllowsChildren() визначає, чи може вузол мати дітей. У нашому випадку дітей не можуть мати вузли останнього рівня, що зберігають дані про викладачів. Тому, щоб не реалізовувати одне й теж саме у класах для зберігання об’єктів предметної області, цей метод можна реалізувати тут, повертаючи у ньому значення true. А у класі для викладача, в разі потреби перевизначити його так, щоб він повертав false.



Методи getChildAt, getChildCount, getIndex об’єднує необхідність циклічної обробки переліку дітей вузла. Метод getChildAt повертає посилання на сина, індекс якого передано як параметр. Метод getChildCount повертає кількість синів. Метод getIndex повертає індекс сина, посилання на якого передано як параметр.

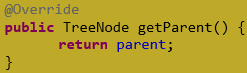
Для реалізації цих методів можна скористатися можливістю, яку надає інтерфейс Iterable, тобто використання циклу for у стилі forEach.

Як приклад наводимо реалізацію методу getIndex:

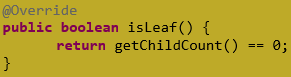


Слід зазначити, що для деяких структур даних ці методи можуть виявитися не дуже ефективними. Врахування особливостей конкретної структури даних може дозволити написати більш ефективний код і перевизначити ці методи у класах спадкоємцях. Можливо це доведеться зробти під час захисту роботи.

Метод getParent повертає посилання на батьківський вузол, яке зберігається у полі parent. Тож метод буде таким:

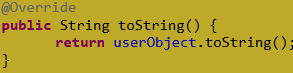


Метод isLeaf() повертає true, якщо вузол є листом. У цьому випадку кількість дітей вузла дорівнює 0. Отже цей метод можна реалізувати через метод getChildCount, що повертає кількість дітей вузла:



#### Реалізація методів інтерфейсу для роботи з даними вузла

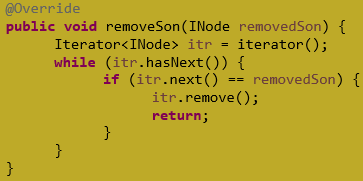
Ці методі були оголошені в інтерфейсі INode. Але перед реалізацією методів інтерфейсу INode перевизначимо метод toString() для вузла. Цей метод буде використовувати компонент JTree для формування рядка, що буде виводитися поруч з позначкою вузла.



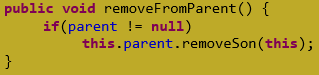
Після цього реалізуємо методи getUserObject() та setUserObject(INode) для поля userObject.

Метод додавання add залежить від способу збереження синів, а це залежить від рівня дерева, на якому відбувається додавання. Тому доведеться реалізувати цей метод у класах спадкоємцях, а у цьому класі цей метод слід вилучити.

Метод вилучення сина вузла removeSon(INode) можна реалізувати через ітератор:



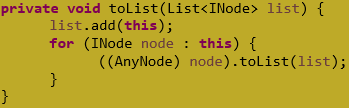
Метод removeFromParent() реалізуємо через метод remove(INode):



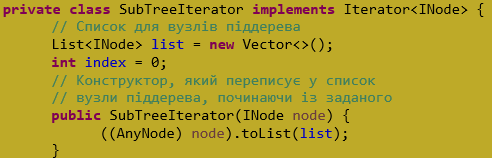
#### Реалізація методу subTreeIterator

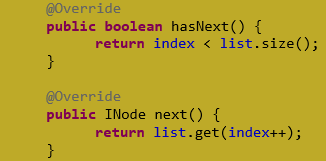
Реалізація цього методу потребує декількох кроків. Спочатку ми реалізуємо приватний метод toList(), який буде переписувати вузли піддерева у список. Потім реалізуємо приватний клас для ітератора піддерева, а вже після цього реалізуємо метод subTreeIterator.

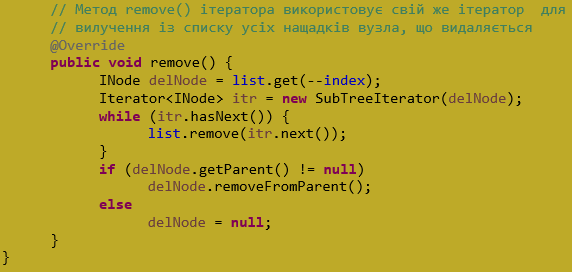
Створімо метод toList(List), у якому вузли піддерева за допомогою рекурсії переписуються до заданого списку типу List:



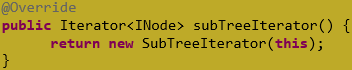
Далі створімо внутрішній клас TreeIterator, який будемо використовувати для створення ітераторів по піддереву.







Тепер можна реалізувати метод subTreeIterator(), у якому ми скористаймося конструктором щойно створеного класу:



На цьому реалізацію класу AnyNode завершено. Слід перевірити, чи не залишилися у класі заготовки нереалізованих методів. Ці методи будуть реалізовуватися у класах спадкоємцях.

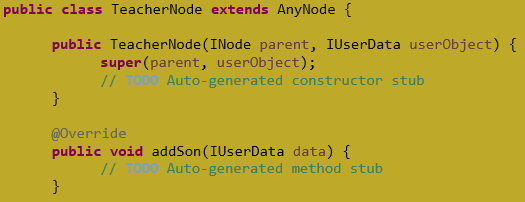
Далі почнемо реалізовувати класи для вузлів на різних рівнях дерева, і почнемо з останнього, бо він не використовує інші класи, а його використовує клас попереднього рівня.

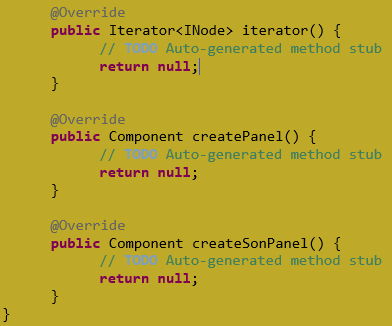
### Класу TeacherNode для збереження даних про викладачів

Створімо клас TeacherNode, що успадковує клас AnyNode і використовує конструктор суперкласу та реалізує абстрактні методи. (Треба включити відповідні прапорці в діалозі створення класу).

Після створення класу ми маємо отримати конструктор і заготовки методів, що потребують реалізації.

Наведемо, як приклад, результат генерування цього класу:





Як бачимо, потрібно реалізувати чотири методи.

Але перед тим, як перейти до їх реалізації, слід визначитися з полями класу. Відповідно до технічного завдання, викладачі мають зберігатися у лінійному однонаправленому списку. То ж у класі має бути поле з посиланням на наступний елемент списку. Визначимо це поле у класі TeacherNode.

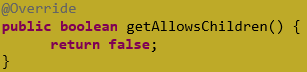


Окрім того, поле має бути доступним об’єктам інших класів, тому для нього слід створити публічні геттер та сеттер:



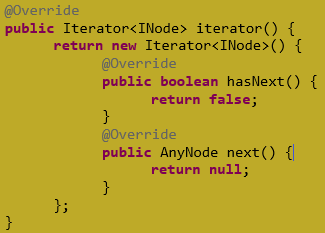


Реалізація методів інтерфейсів цього класу не є складною тому, що це клас останнього рівня і не може мати дітей. Для того, щоб визначити цю обставину слід перевизначити у класі метод getAllowsChildren()

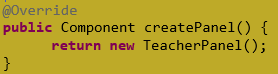


Конструктор і метод addSon(IUserData) не потребують змін. Додаватися до вузлів цього типу нічого не буде.

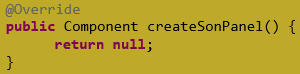
Ітератор доведеться створити, але такий, що нічого не буде робити:



Метод, що створює панель для введення даних про викладача, має такий вигляд:



А метод, що має повертати панель для введення інформації про синв вузла залишається без змін, тобто повертає null:



### Реалізація класу DeptNode

Створюємо цей клас так само як і попередній. Він має успадковувати клас AnyNode, використовувати конструктор суперкласу і реалізувати абстрактні методи інтерфейсів.

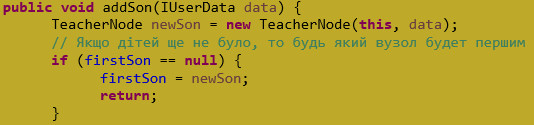
Відповідно до технічного завдання цей клас зберігає своїх нащадків у лінійному однонаправленому списку, тож у класі має бути визначено поле для посилання на перший елемент цього списку з даними типу TeacherNode (вузол для першого викладача):

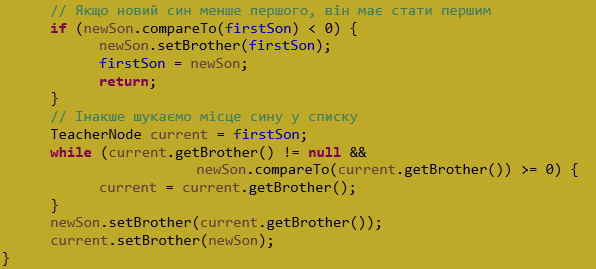


Перейдемо до реалізації методів інтерфейсів.

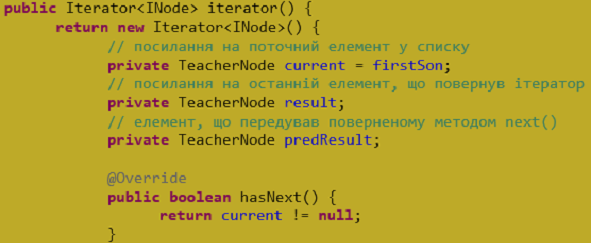
Метод addSon(IUserData) має враховувати вимогу, що масив має бути впорядкованим.

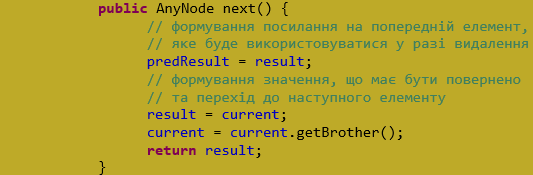
Нижче наведено реалізацію цього методу:

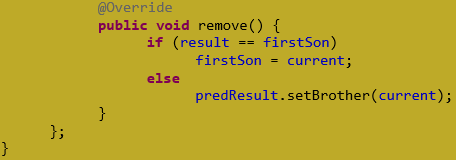




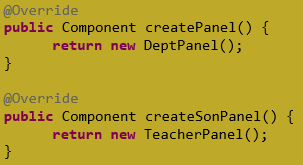
Ітератор реалізуємо через анонімний об’єкт анонімного класу:







Методи для створення інформаційних панелей будуть виглядати так:



### Реалізація класу FacultyNode

Цей клас слід створити так само як і попередній. Він успадковує клас AnyNode, використовує конструктор суперкласу і реалізує абстрактні методи інтерфейсів.

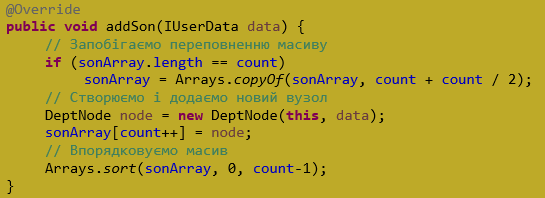
Відповідно до технічного завдання цей клас зберігає своїх нащадків у масиві, тож у класі має бути визначено поле для посилання на масив типу DeptNode (масив вузлів для кафедр) і поле для лічильника нащадків у масиві:



Реалізуємо абстрактні методи інтерфейсів у цьому класі.

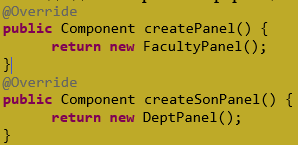
У методі addSon(IUserData) необхідно контролювати розмір масиву і своєчасно розширювати його. Окрім того, масив має бути впорядкованим.

Нижче наведено реалізацію цього методу:

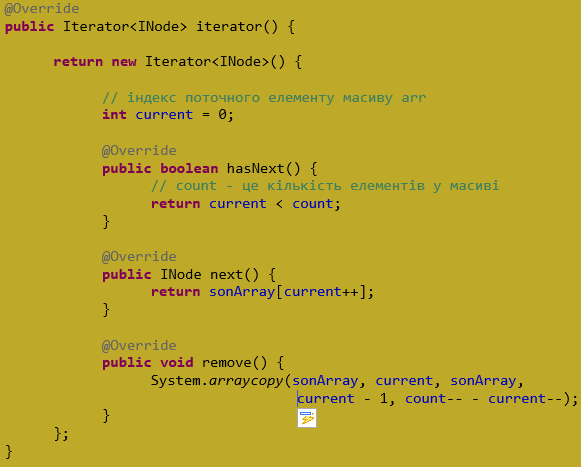


Впорядкування масиву за допомогою методу sort утилітного класу Arrays не буде суттєво уповільнювати метод. У Java8 використовується алгоритм сортування, що ефективно використовує вже відсортовані частини масиву.

Методи для створення інформаційних панелей будуть виглядати так:



Ітератор реалізуємо у вигляді анонімного об’єкту анонімного класу:



### Клас UniverNode для кореневого вузла

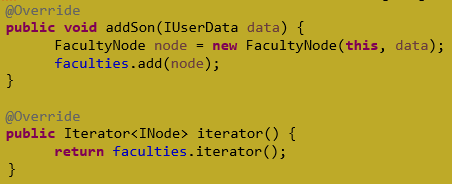
Створімо клас UniverNode, що успадковує клас AnyNode, використовує конструктор суперкласу і реалізує абстрактні методи.

Після створення класу ми маємо отримати конструктор і заготовки методів, що потребують реалізації.

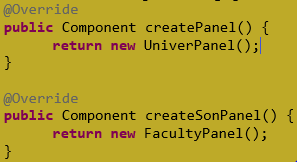
Але перед тим, як перейти до їх реалізації, слід визначитися із полями класу. Відповідно до технічного завдання, факультети треба зберігати у колекції TreeSet, то й визначимо у класі таке поле:



Реалізація методів інтерфейсів цього класу не є складною, тому що клас TreeSet має метод для додавання елементів і має ітератор:



Методи для створення інформаційних панелей будуть виглядати так:



### Реалізація моделі дерева

Маючи класи для вузлів дерева можна реалізовувати усі операції з деревом, тому що корінь ідентифікує усе дерево.

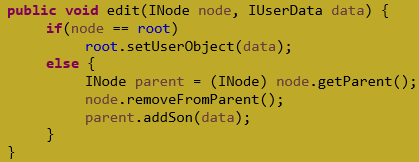
Але інколи може знадобитися об’єкт, що представляє модель дерева. Тому ми також зробимо клас моделі дерева, що реалізує інтерфейсTreeModel. За основу можна взяти клас DefaultTreeModel, що реалізує інтерфейс TreeModel і базується на інтерфейсі TreeNode.

Створімо клас ModelForTree та продублюємо конструктор суперкласу, через який буде визначатися корінь.



Створімо також методи для маніпуляцій з деревом:







## Програмування візуальної частин

### Реалізація методу onCreateRoot()

Цей метод ми вже частково створювали, коли тестували панелі. Та завдання цього методу полягає не тільки у тому , щоб відобразити інформаційну панель, але й у створені кореня дерева на підставі отриманих даних. Створення кореня має відбуватися після натискання на кнопку «Save», але у загальному випадку кнопка не знає, з якою метою відкривалася панель. Тож створювати корінь дерева має метод onCreateRoot. І проблема тут полягає у тому, що виконання коду методу після відкриття панелі має бути затримано до натискання кнопки.

Щоб вирішити цю проблему, ми маємо створити у методі onCreateRoot() нитку(thread) виконання коду і скористатися яким небудь засобом для призупинення потоку та його відновлення. Таким засобом може бути об’єкт latch класу CountDownLatch.

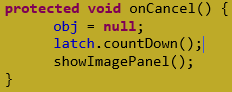
Друга проблема полягає у тому що у нас ще є кнопка «Cancel». Натискання на цю кнопку відміняє операцію. Але метод onCreateRoot не знає, яку кнопку натиснули.

Для вирішення другої проблеми ми визначимо у класі поле obj типу IUserData. Кнопка «Cancel» буде записувати null у це поле, а кнопка «Save» буде записувати об’єкт, створений з використанням даних панелі. Далі кнопки мають активізувати продовження виконання коду методу onCreateRoot().

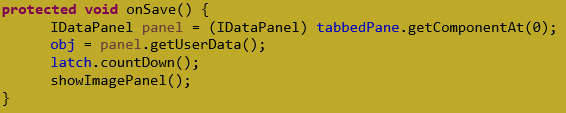
Оголосимо ці поля:



Реалізуємо метод onCancel():

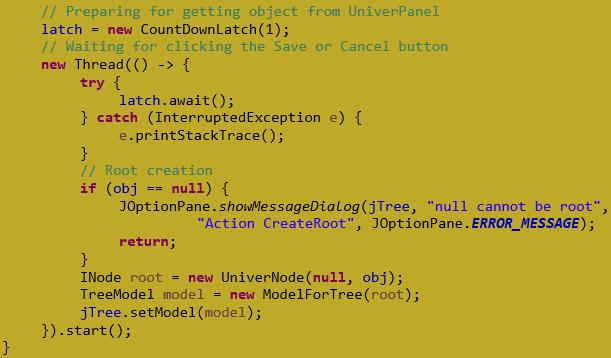


Реалізуємо метод onSave():



Реалізуємо метод onCreateRoot()

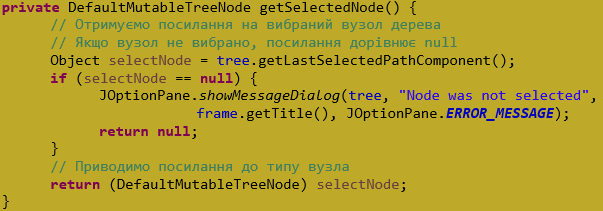




### Реалізація методу onAdd()

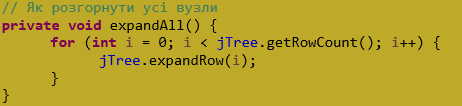
Цей метод має додавати синів до вибраного вузла. І перша проблема яка тут виникає – як отримати доступ до вибраного вузла?

Для спрощення цієї проблеми створімо допоміжний метод getSelectedNode():



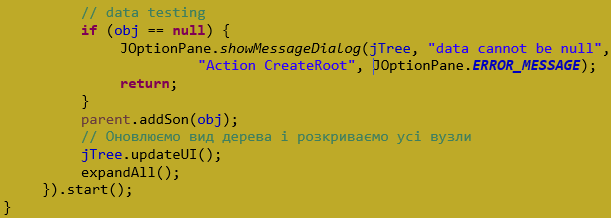
Ще одна невелика проблема додавання полягає у тому, що компонент JTree не розкриває вузол, до якого додано сина. Це ускладнює контроль за виконанням операції.

Тому створімо допоміжний метод, що розкриває усі вузли дерева:



Тепер можна реалізувати метод додавання вузла onAdd(), який багато в чому схожий на метод onCreateNode()

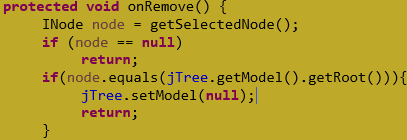


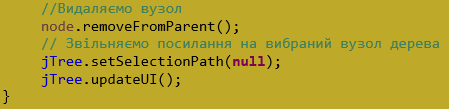


Тепер можна тестувати роботу з деревом, додаючи вузли на усіх рівнях.

### Реалізація операції видалення вузла

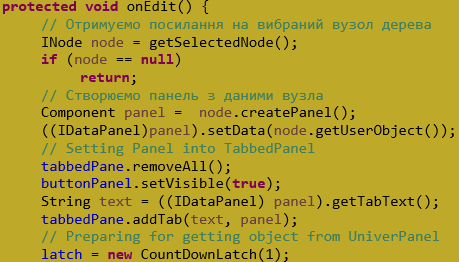
Далі слід реалізувати метод onRemove(), і протестувати його.

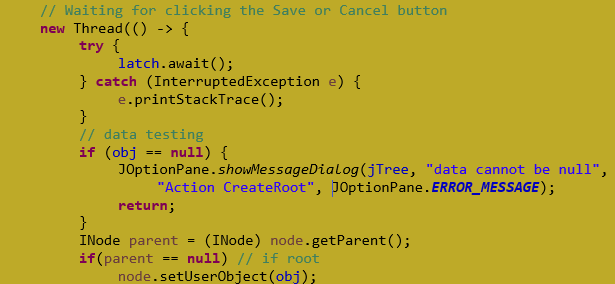


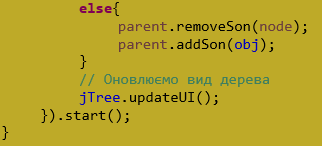


### Реалізація операції редагування даних вузла

Реалізуємо метод onEdit():



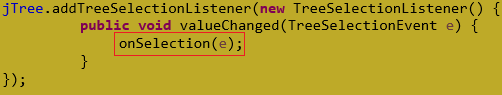




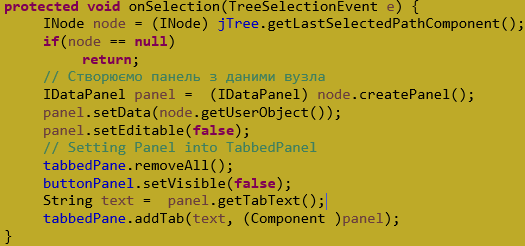
### Реалізація перегляду змісту вузлів

Для реалізації перегляду змісту вузла, який було виділено, будемо використовувати подію treeSelection компоненту JTree.

Створімо обробника події treeSelection для компонента JTree з такою реалізацією методу обробки події, рамочкою виділено текст, який треба додати до заготовки, що згенерована Eclipse:



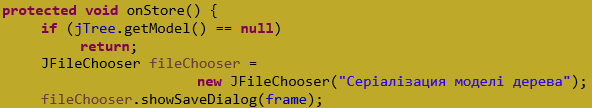
Після цього реалізуємо protected метод onSelection:

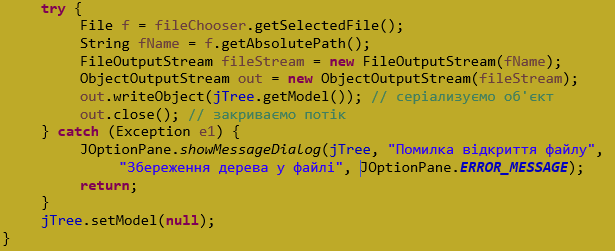


Тепер можна протестувати роботу діалогів та даних вузлів у режимі перегляду.

### Реалізація операції збереження дерева у файлі

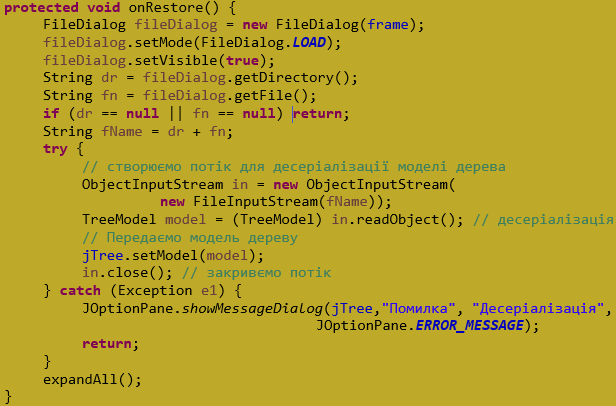
Для виконання цієї операції реалізуємо метод onStore:





### Реалізація відновлення дерева з файлу

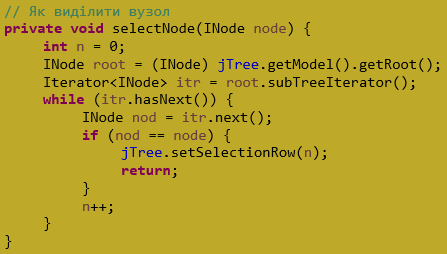
Для виконання цієї реалізуємо метод onRestore:



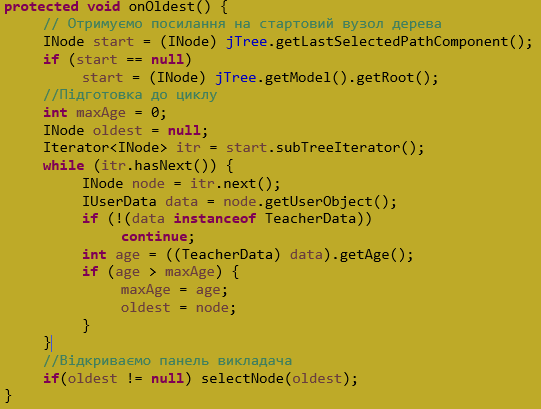
### Пошук викладача, найстаршого за віком

Це не складне завдання. Можна скористатися ітератором, який надає метод subTreeIterator() і, обходячи усі вузли, аналізувати вузли типу TeacherNode.

Але знайшовши відповідний вузол, ми маємо його виділити на дереві. Для того, щоб це зробити, напишемо допоміжний метод:



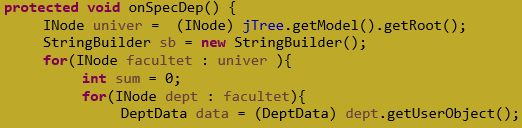
Тепер можна реалізувати метод onOldest():

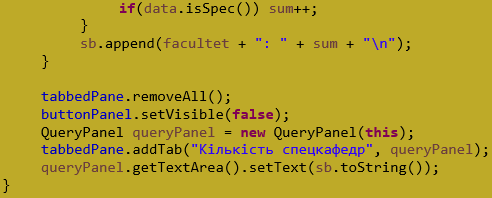


### Підрахунок кількості випускаючих кафедр на факультетах

Реалізуємо метод onSpecDep(). Результати пошуку будемо виводити на панель Query Panel.

Нижче наведено текст методу:





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 МартинАрхитектура корпоративных программных приложений. СПб.: Издательский дом «Вильямс», 2007. – 544с
6. Simon Kendal. Object oriented programming using Java. Ventus Publishing ApS-, 2009. – 209 c.
7. http://omg.org Object Management Group