**ЛАБОРАТОРНА РОБОТА № 9**

з дисципліни

«Розробка програмного забезпечення на Java»

Шейко Р. О.

Групи КІ-21-2

**Завдання на лабораторну роботу**

1. Прочитайте теоретичну частину;

2. Зробіть усі приклади з теоретичної частини;

3. Дайте відповідь на контрольні питання.

1. Прочитав теоретичну частину. Код знаходиться у завданні, тому додатків в цій лабораторній роботі не буду.
2. Протестуємо всі додатки:

**Першим додатком** буде простий зв`язний список. Цей різновид списку підтримує такі дії:

- Вставка елемента напочатку списку;

- Видалення елемента напочатку списку;

- Перебір списку для виведення вмісту

Код до нього знаходиться на стр 3, 4, 6(Лістинг 1.1). Ось приклад тестування:

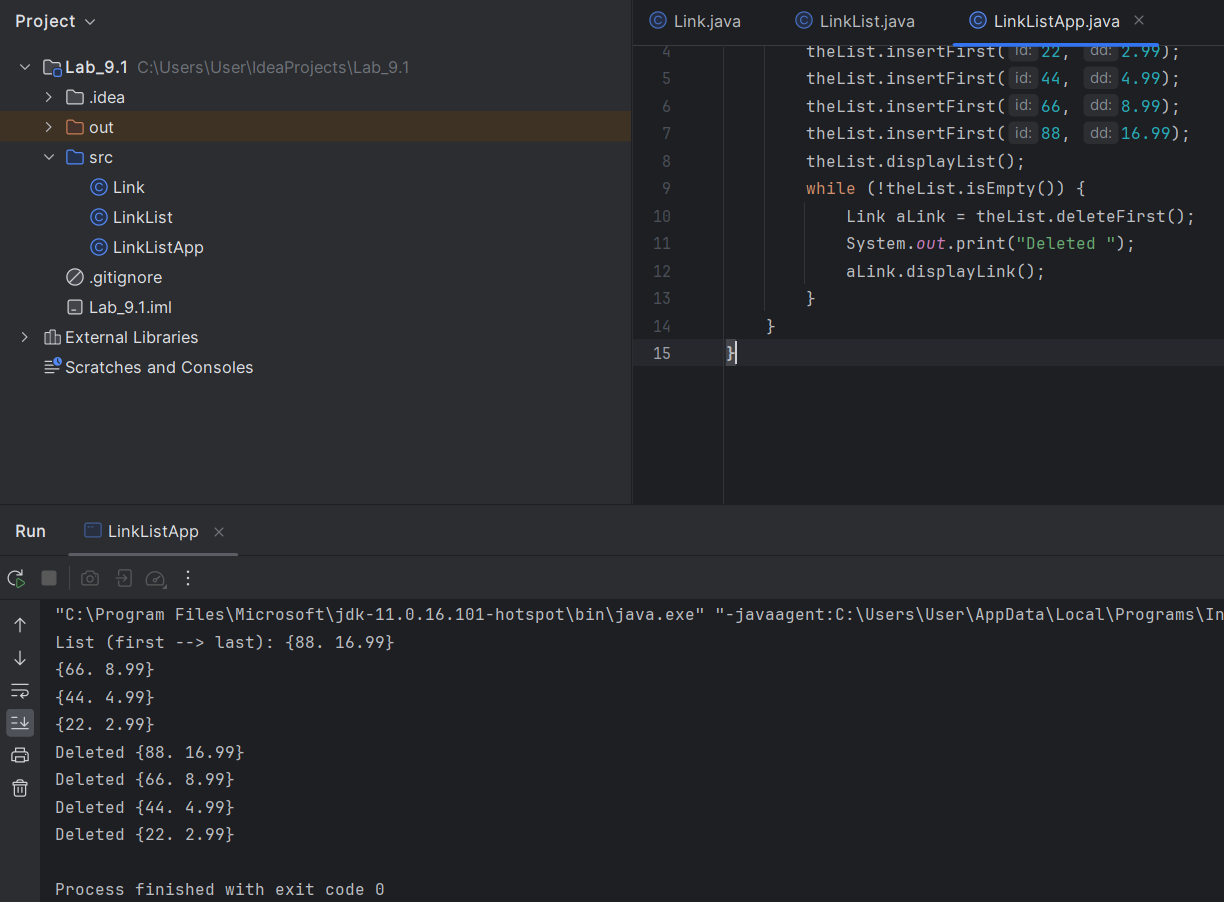


Рис 1. Тестування простого зв`язного списку

**Другий додаток** доповнений методами для пошуку та видалення елементів. Додані функції знаходяться у Лістингу 1.2. Для пошуку елемента, що видаляється, метод delete() використовує той же спосіб, що і метод find(). Але цей метод повинен зберігати посилання не тільки на поточний елемент списку, а й на попередній, це необхідно через те, що при видаленні поточного елемента метод повинен пов'язати попередній елемент з наступним (рис. 1.4). Щоб звернутись до попереднього елемента, необхідно зберегти посилання на нього.

Протестуємо:

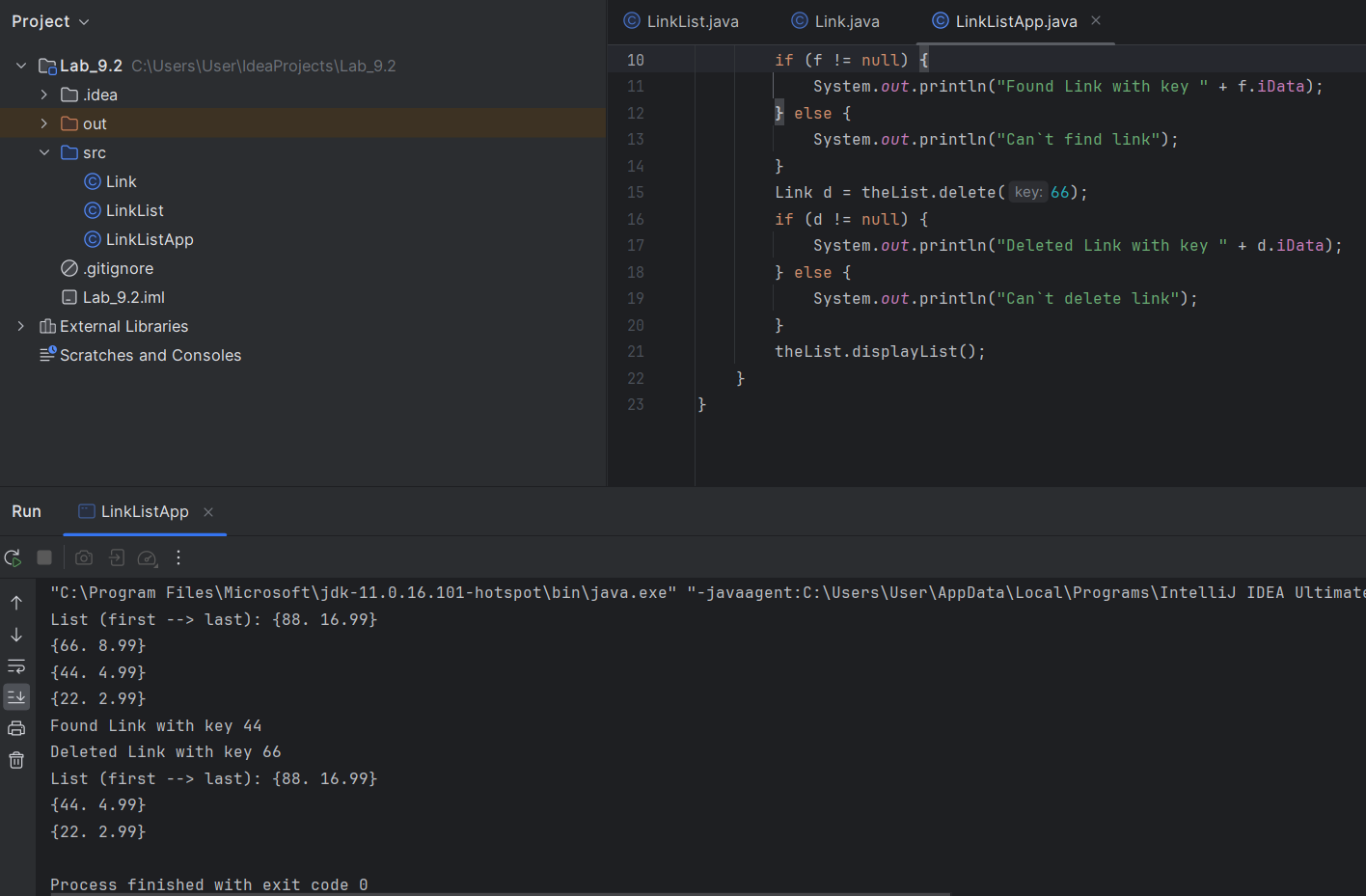


Рис 2. Тестування другого додатку

Створимо **третій додаток з** двостороннім списком. Двосторонній список схожий на звичайний пов'язаний список з однією додатковою можливістю: у ньому зберігається посилання як на перший елемент, а й у останній. Код знаходиться у лістингу 1.3. Тестування зображене нижче:

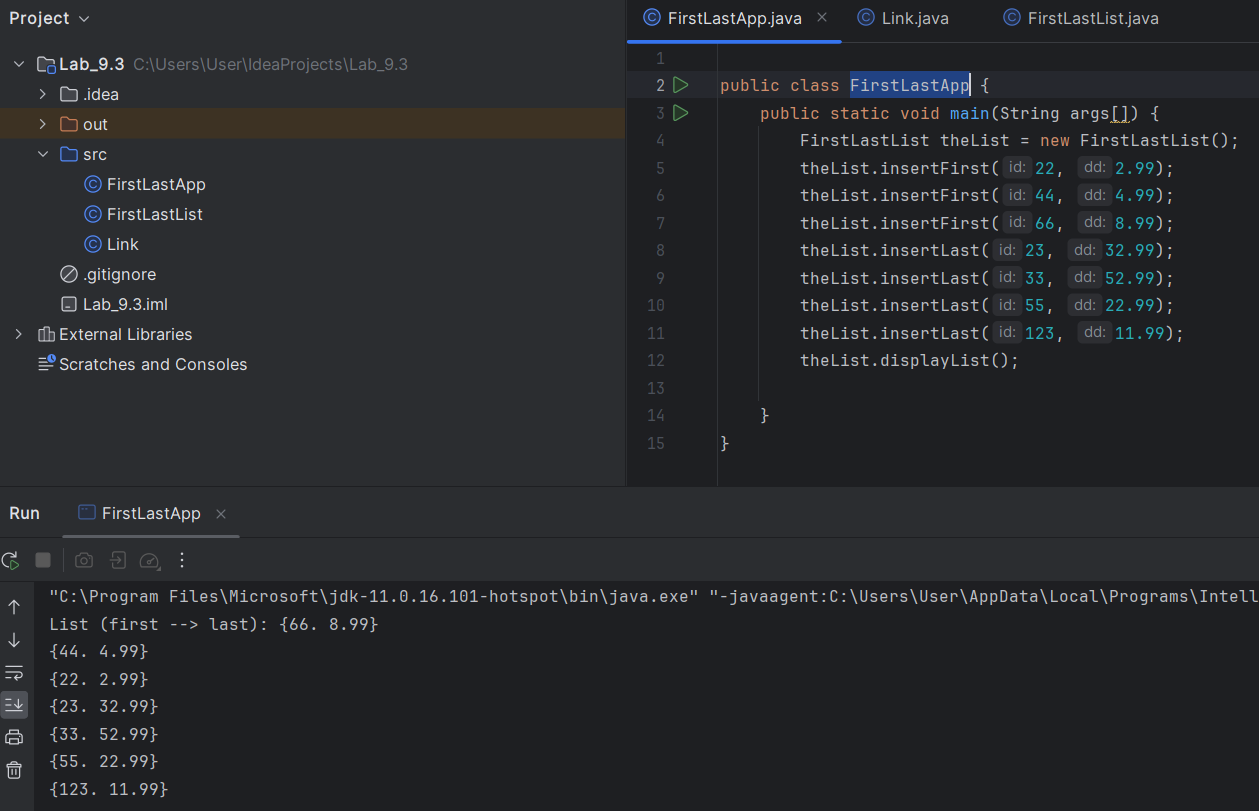


Рис 3. Тестування третього додатку

Створимо **четвертий додаток з** стеку на базі списку. У лістингу 3.1 надано можливу реалізацію класу стека LinkStack, в якій замість масиву використовується клас LinkList. Тестування зображене нижче:

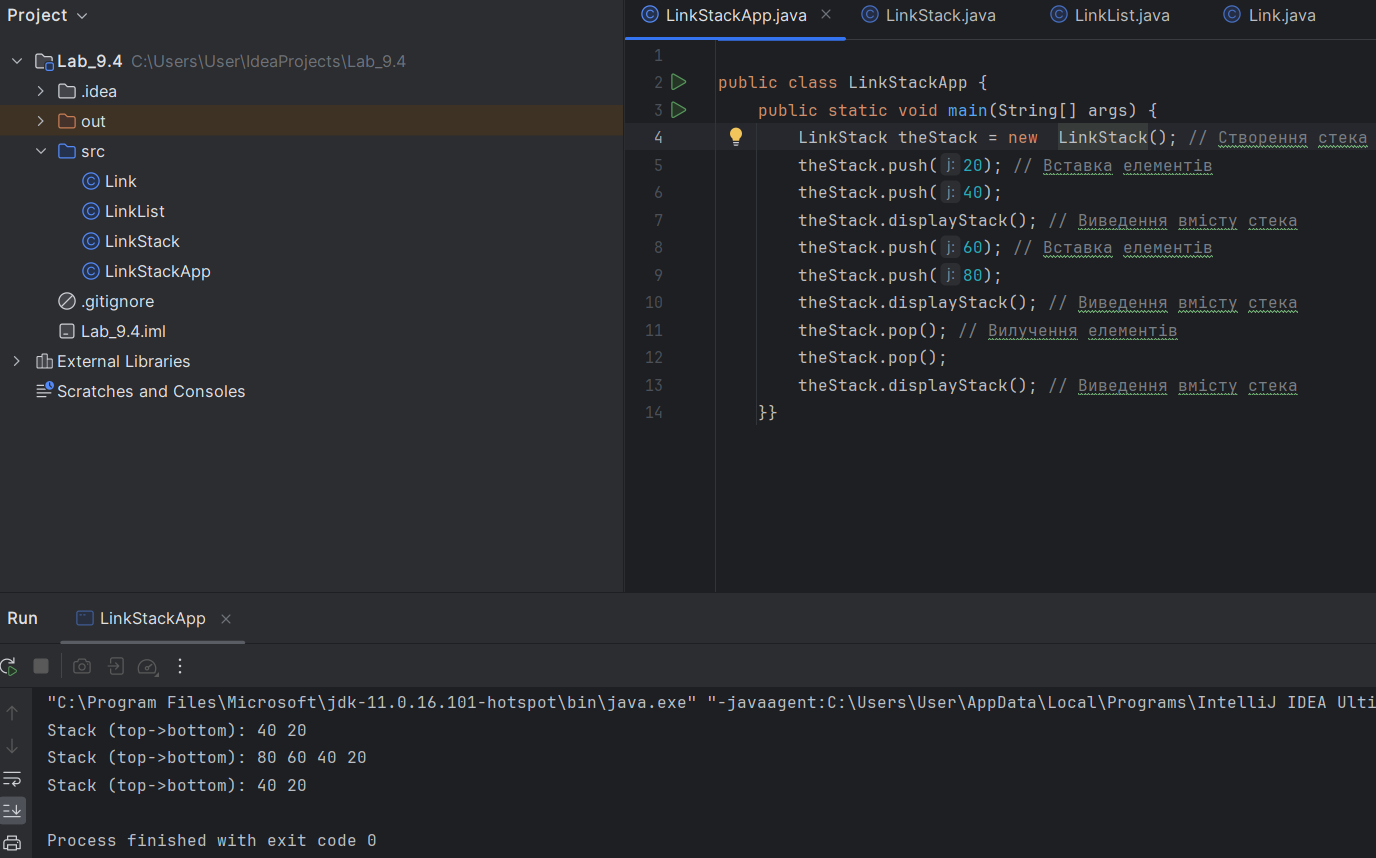


Рис 4. Тестування четвертого додатку

Створимо **п`ятий додаток з** linkQueue. Розглянемо інший приклад реалізації ADT. Програми linkStack.java і linkQueue.java наочно показують, що стеки і черги є концептуальні сутності, відокремлені від їх реалізацій. Стек може бути з рівним успіхом реалізований як на базі масиву, так і на основі пов'язаного списку. Для роботи стека важливими є операції push()/pop() і те, як вони використовуються, а не базовий механізм, обраний для реалізації цих операцій. У лістингу 3.2 представлена черга, реалізована на базі двостороннього зв'язаного списку. Тестування зображене нижче:

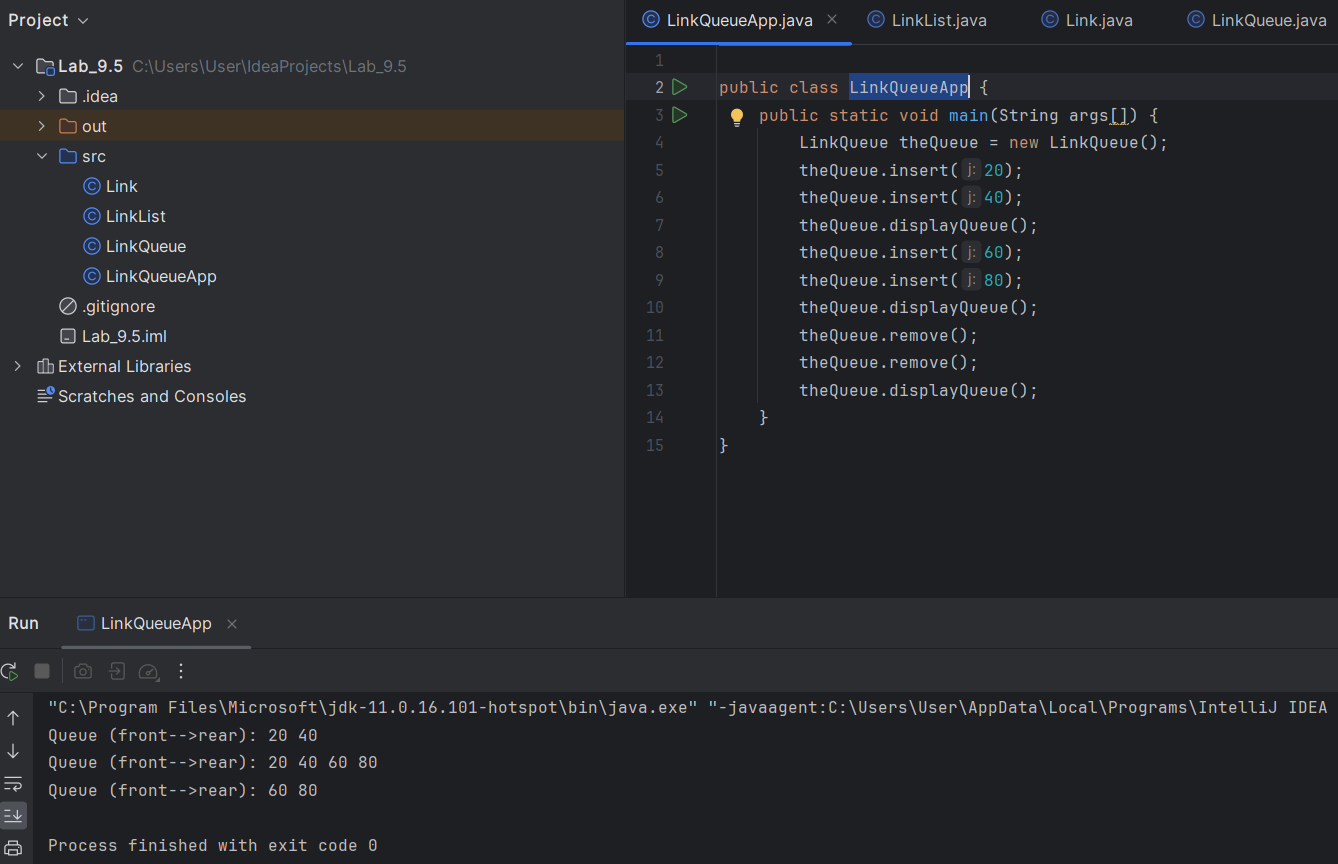


Рис 5. Тестування п`ятого додатку

Створимо **шостий додаток з** сортованого списку. Вставка та видалення довільних елементів у сортованих зв'язаних списках вимагають O(N) порівнянь (у середньому N/2), тому що позицію виконання операції доводиться шукати перебором списку. З іншого боку, пошук або видалення найменшого значення виконується за час O(1), тому що воно завжди знаходиться на початку списку. Якщо програма часто звертається до найменшого елемента, а швидкість вставки не критична, то пов'язаний сортований список буде досить ефективним. Наприклад, пріоритетна черга може бути реалізована на базі сортованого зв'язаного списку. Тестування зображене нижче:

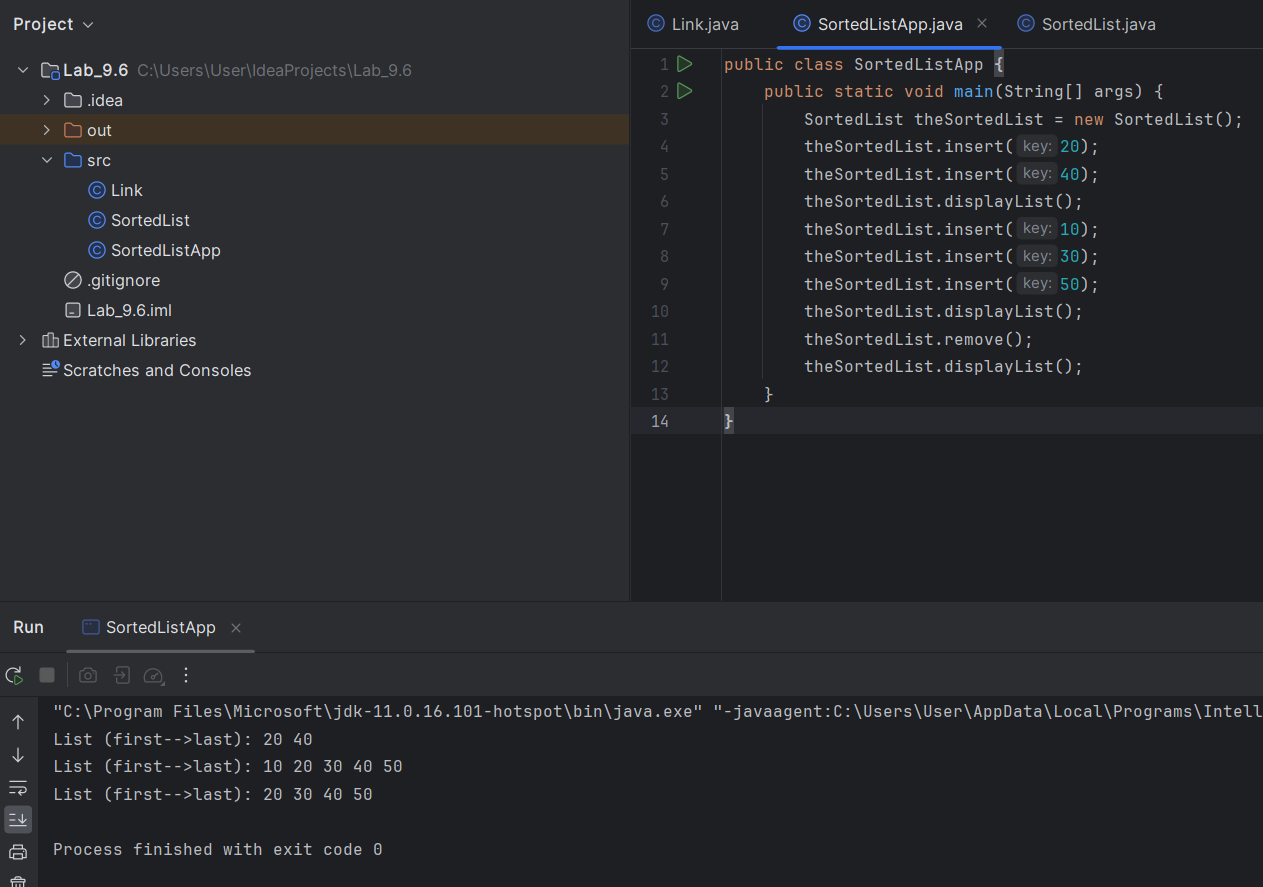


Рис 6. Тестування шостого додатку

Створимо **сьомий додаток з** сортованого списку методом вставки. Недолік сортування методом вставки в список (порівняно з сортуванням методом вставки в масив) полягає в тому, що вона займає вдвічі більше пам'яті, оскільки масив та пов'язаний список повинні бути в пам'яті одночасно. І все ж, якщо у вас є готовий клас сортованого списку, такий спосіб сортування добре підходить для невеликих масивів. Код знаходиться у лістингу 4.2. Тестування зображене нижче:

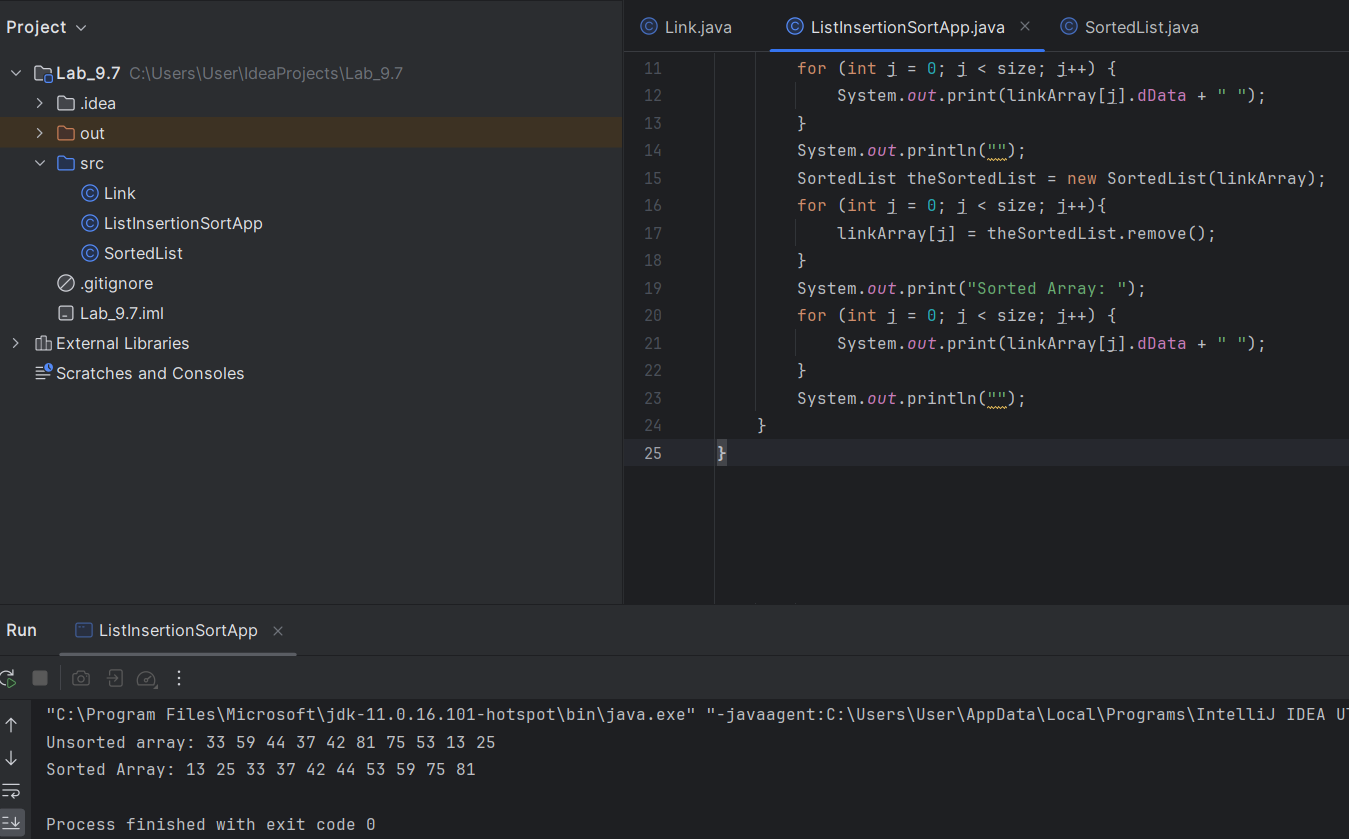


Рис 7. Тестування сьомого додатку

Створимо **восьмий додаток з** двозв`язного списку. Двозв'язковий список може використовуватися як база для побудови дека, або двосторонньої черги. У деці операції вставки та видалення можуть виконуватися з обох кінців; двозв'язковий список надає таку можливість. Код знаходиться у лістингу 5.1. Тестування зображене нижче:

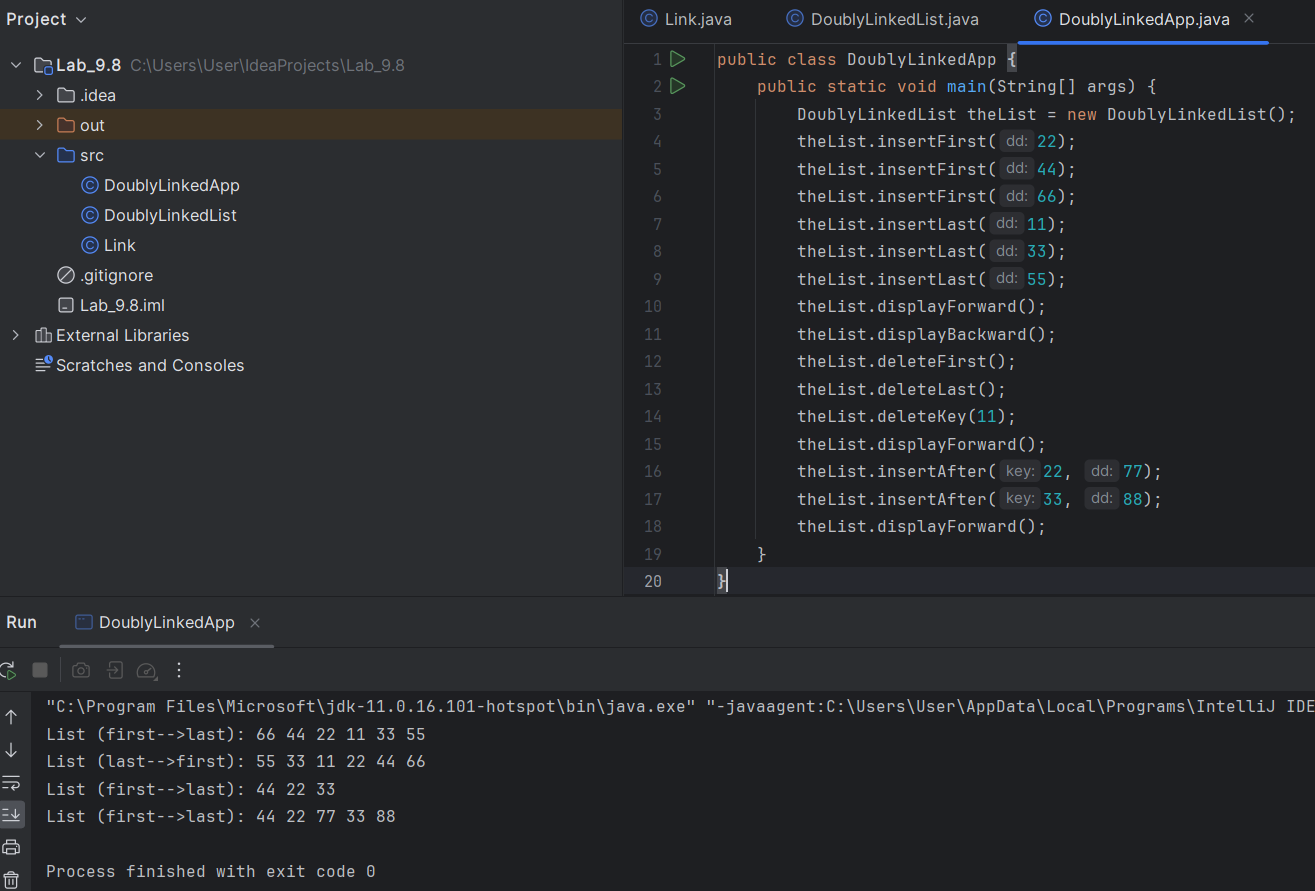


Рис 8. Тестування восьмого додатку

Створимо **дев`ятий додаток з** ітератором Об'єкти, що містять посилання на елементи структур даних і використовувані для перебору цих елементів, зазвичай називаються ітераторами. Визначення класу ітератора виглядає приблизно так:

class ListIterator { private Link current; }

Поле current містить посилання на елемент, який в даний час вказує ітератор. Для використання ітератора користувач спочатку створює список, а потім об'єкт-ітератор, асоційований з цим списком. Насправді простіше доручити створення ітератора списку, тому що він може передати ітератору корисну інформацію, скажімо, посилання на першу. Завдяки інтерактивному інтерфейсу було легше розібратися, що таке ітератор. Код знаходиться у лістингу 6.1. Тестування зображене нижче:

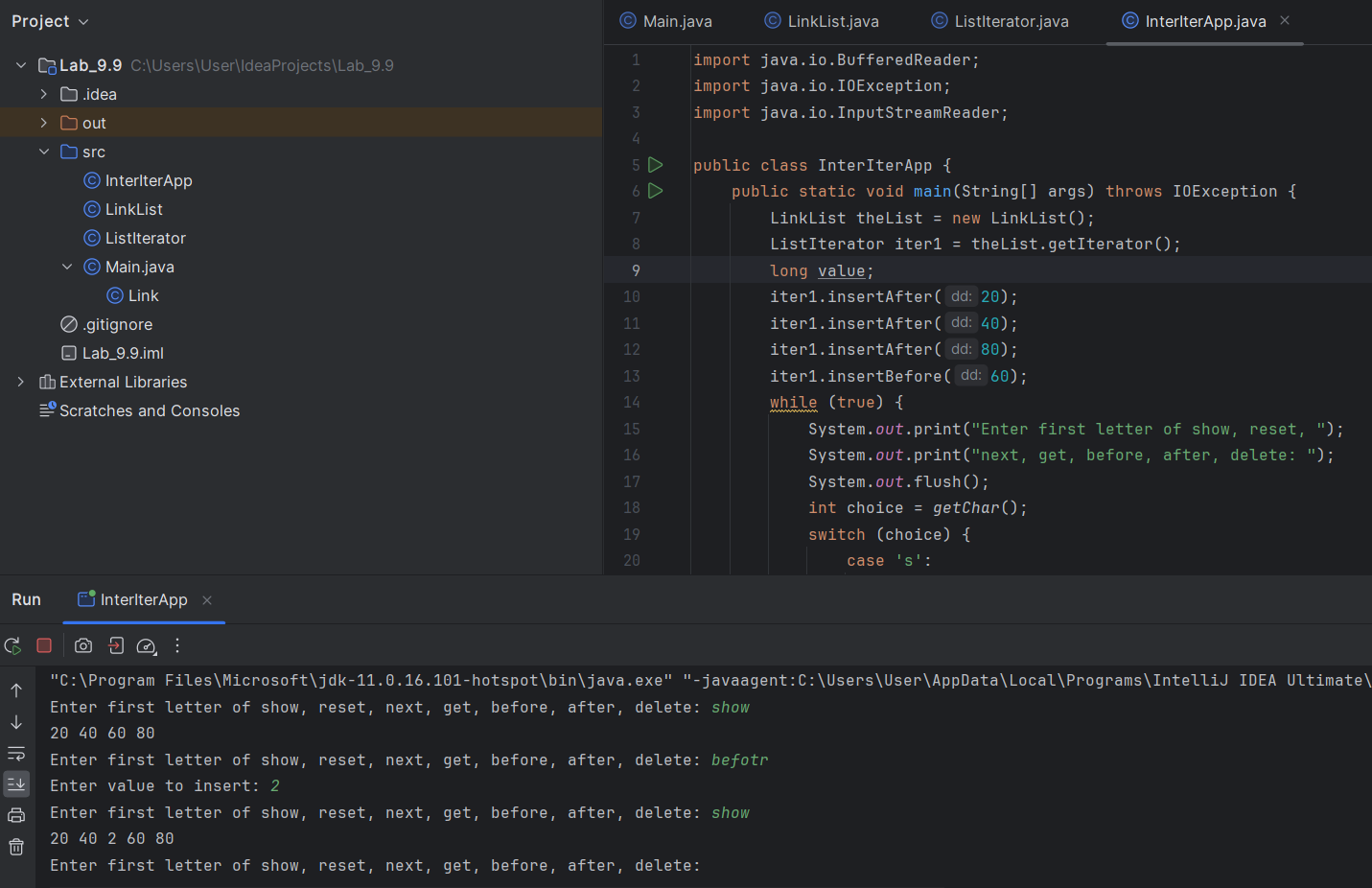


Рис 8. Тестування дев`ятого додатку

**3. Відповіді на питання**

1. Пов'язаний список - це структура даних, яка складається з вузлів (елементів), де кожен вузол містить дані та посилання на наступний вузол в списку. Принцип роботи полягає в тому, що дані зберігаються у вузлах, які з'єднані ланцюжком. Кожен вузол має вказівник на наступний вузол, і це дозволяє легко додавати, видаляти та змінювати елементи в списку.

2. Пошук та видалення в простому зв'язаному списку відбувається наступним чином:

- Пошук: Починаючи з першого вузла (голови), перевіряємо, чи вміст поточного вузла відповідає шуканому значенню. Якщо так, то знаємо, що ми знайшли потрібний вузол. Якщо ні, переходимо до наступного вузла та продовжуємо пошук.

- Видалення: Щоб видалити вузол зі списку, ми змінюємо вказівники в попередньому вузлі, щоб вони "обходили" вузол, який ми хочемо видалити. Після цього вузол може бути звільнений з пам'яті.

3. Двосторонні списки (або двозв'язкові списки) відрізняються від звичайних (односторонніх) списків тим, що в кожному вузлі зберігаються посилання як на наступний, так і на попередній вузол у списку. Це дозволяє легко перейти в обидві сторони від поточного вузла і спрощує деякі операції, такі як вставка та видалення елементів у середині списку.

4. Абстрактні типи пам'яті - це абстрактні концепції, які визначають, як дані зберігаються та використовуються в пам'яті комп'ютера. Вони включають такі типи, як стеки, черги, масиви, списки та інші, і визначають правила доступу та операції, які можна виконувати з даними.

5. За допомогою пов'язаного списку можна реалізувати різні структури даних, включаючи звичайні списки, стеки, черги, черги з пріоритетами, асоціативні масиви та інші. Відмінність від реалізації на основі масиву полягає в тому, що пов'язані списки динамічні, тобто розмір списку може змінюватися під час виконання програми, тоді як масиви мають фіксований розмір.

6. Сортовані списки - це списки, в яких елементи зберігаються в відсортованому порядку. Ефективність сортованих списків залежить від конкретної реалізації та використаної алгоритмічної операції для вставки та пошуку. Якщо список завжди відсортований, то пошук може бути виконаний швидше (наприклад, за допомогою бінарного пошуку), але вставка може бути більш часомісткою операцією.

7. Прості списки мають посилання лише на наступний вузол, тоді як двозв'язкові списки мають посилання як на наступний, так і на попередній вузол. Це дозволяє легко перейти в обидві сторони від поточного вузла у двозв'язковому списку, що спрощує деякі операції, такі як видалення в середині списку.

8. Ітератори - це об'єкти або конструкції мови програмування, які дозволяють послідовно обходити елементи в структурі даних. Вони корисні для доступу до даних у списках, масивах та інших структурах даних безпосередньо і дозволяють виконувати операції, такі як ітерація чи пошук без необхідності вручну зберігати поточне положення в структурі даних.

**Висновок**

У ході виконання лабораторної роботи ми дослідили та розглянули різні аспекти структур даних та абстрактних типів пам'яті. Основні точки, які були розглянуті та відповіді на контрольні запитання, включають:

1. Ми зрозуміли, що пов'язаний список - це структура даних, де кожен елемент має посилання на наступний елемент у списку. Принцип роботи полягає в послідовному переході від одного елементу до іншого, використовуючи посилання.

2. Пошук та видалення в простому зв'язаному списку вимагає послідовного перегляду вузлів, починаючи з початку списку, та зміни вказівників для видалення вузла.

3. Двосторонні списки відрізняються від звичайних тим, що мають посилання як на наступний, так і на попередній елемент, що дозволяє зручно працювати зі списками в обидві сторони.

4. Абстрактні типи пам'яті - це абстрактні концепції, які визначають, як дані зберігаються та використовуються в пам'яті комп'ютера.

5. За допомогою пов'язаного списку можна реалізувати різні структури даних, такі як звичайний список, стек, черга, черга з пріоритетами, асоціативний масив та інші. Відмінність від реалізації на основі масиву полягає в тому, що пов'язані списки є динамічними і дозволяють змінювати розмір списку під час виконання програми.

6. Сортовані списки можуть бути ефективними, але ефективність залежить від конкретної реалізації та алгоритмів сортування, які використовуються для додавання нових елементів.

7. Основна відмінність між простими та двозв'язковими списками полягає в тому, що двозв'язкові списки мають додаткове посилання на попередній елемент, що дозволяє зручно працювати зі списками в обидві сторони.

8. Ітератори - це корисні інструменти для послідовного обходу елементів у структурах даних, дозволяючи виконувати операції безпосередньо над елементами та пам'ятати поточне положення в структурі даних.

У цій лабораторній роботі ми вивчили основи структур даних, їхні принципи роботи та можливості реалізації різних структур з використанням пов'язаних списків. Розуміння цих концепцій є важливим для програмістів і допомагає покращити навички розробки та оптимізації програм.