ПРАКТИЧНА РОБОТА 5
Колекції та класичний ввід-вивід
Система оцінювання

№	Тема	К-ть балів
1.	*Захист принаймні одного завдання з роботи*	1
2.	Завдання на практичну роботу	3,8*
3.	*Здача звіту*	0,2
	Всього	**5**

* – діє бонусна система

1. *0,2 бала* Створіть список (спробуйте як ArrayList, так і LinkedList) та заповніть його назвами країн, введених з клавіатури. Відсортуйте список та виведіть його на екран. Далі кілька разів застосуйте до списку метод Collections.shuffle() та після кожного застосування виведіть результат.

2. *0,3 бала* Напишіть статичний метод reverseCopy(), який має аргумент List<T> і повертає власноруч створену копію List<T> з елементами в зворотному порядку (без використання вбудованого методу).

3. *0,3 бали* (Сортування слів за допомогою TreeSet) Напишіть програму, яка використовує метод String.split() для відокремлення слів з введеного користувачем тексту, а потім розміщує їх у TreeSet. Виведіть елементи TreeSet на екран.

4. *0,4 бала* (Хешовані структури) Реалізуйте додаток, який буде виконувати роль телефонної книги. Кожний запис телефонної книги містить номер телефону та назву контакту. Користувачеві мають бути доступними наступні можливості: додати контакт, видалити контакт, редагувати контакт, вивести список усіх контактів та вийти з додатку. Телефонна книга повинна мати при запуску контакти екстрених служб: «Пожежна безпека» - 101, «Поліція» - 102, «Швидка допомога» - 103. Книгу рекомендується тримати у вигляді мепу, а також в процесі реалізації видалення контакту додати перевірку на непорожність книги. Видалення контакту слід виконувати за назвою контакту.

5. *0,5 бала* *(Створення плейлісту на базі списку)* Створіть плейліст, кожна пісня якого має назву, виконавця, альбом та тривалість. Плейліст дозволяє додавати та видаляти пісню, отримувати розмір та очищати його, а також виконувати форматований вивід композицій і повну тривалість його відтворення.

6. *0,6 бала* (Стиснення Хаффмана) Алгоритм Хаффмана – адаптивний жадібний алгоритм оптимального префіксного кодування алфавіту з мінімальною надмірністю. Це

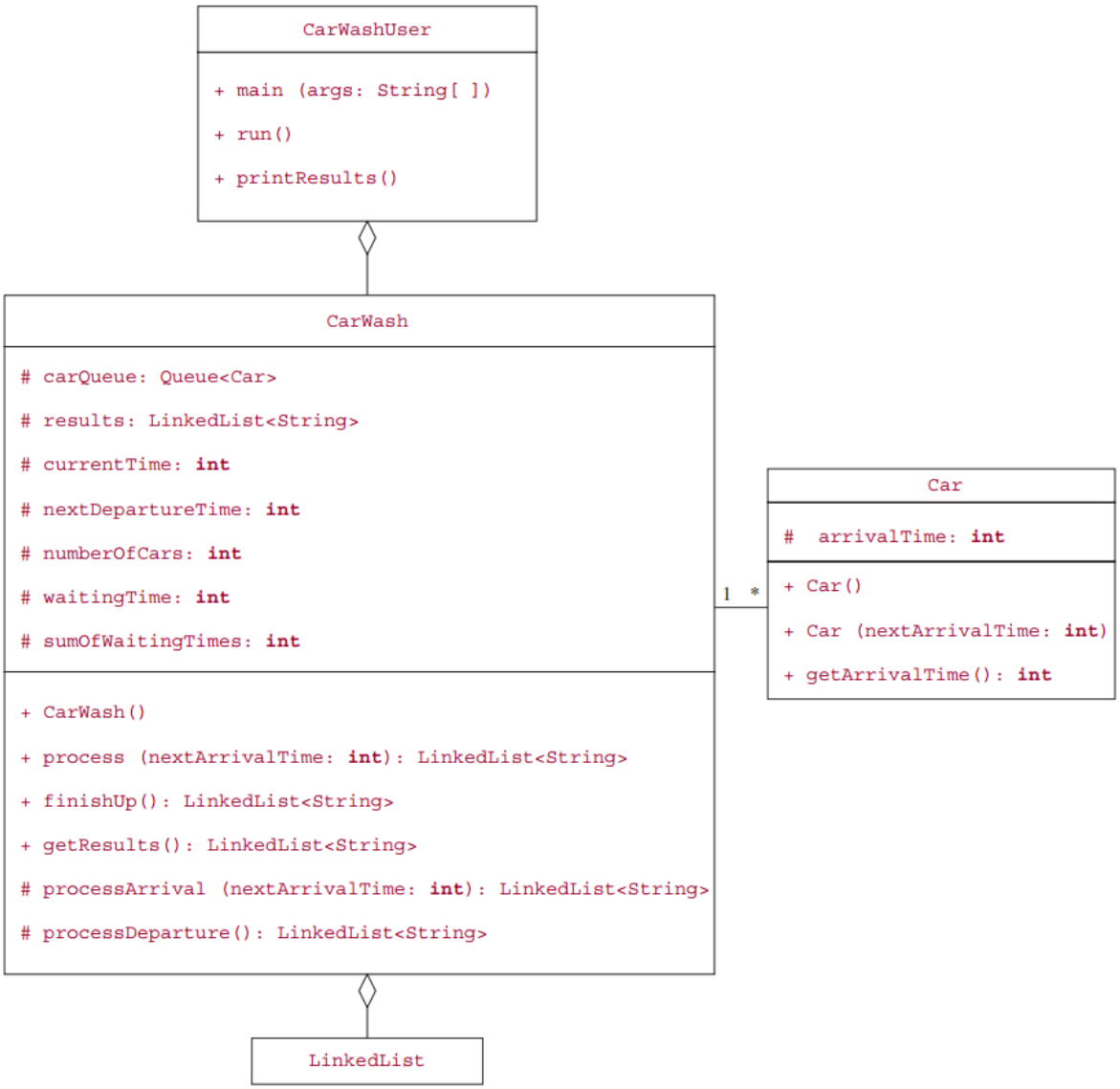
метод присвоєння символам бінарних кодів, що зменшує загальну кількість бітів інформації, за рахунок чого і досягається стиснення.
Елементи дерева Хаффмана (вузли та листки) описуються класами так:

```
abstract class HuffmanTree implements Comparable<HuffmanTree> {
    public final int frequency; // the frequency of this tree
    public HuffmanTree(int freq) { frequency = freq; }

    // compares on the frequency
    public int compareTo(HuffmanTree tree) {
        return frequency - tree.frequency;
    }
}
class HuffmanLeaf extends HuffmanTree {
    public final char value; // the character this leaf represents

    public HuffmanLeaf(int freq, char val) {
        super(freq);
        value = val;
    }
}

class HuffmanNode extends HuffmanTree {
    public final HuffmanTree left, right; // subtrees

    public HuffmanNode(HuffmanTree l, HuffmanTree r) {
        super(l.frequency + r.frequency);
        left = l;
        right = r;
    }
}
```

У даному завданні потрібно сформувати частоти зустрічності кожного символу з тексту, створити деревовидну структуру, а потім виконати обхід дерева. Побудова дерева відбувається на основі черги з пріоритетами. Спочатку матимемо ліс із листків (для кожного символу з тексту).
У результаті роботи програми відповідно до введеного тексту потрібно вивести таблицю символів, що зустрічаються в цьому тексті, їх частоти зустрічності (кількість згадувань у тексті), а також сформовані бінарні коди. Така таблиця для тексту «this is an example for huffman encoding» може виглядати так:

SYMBOL	WEIGHT	HUFFMAN CODE
d	1	00000
t	1	00001
h	2	0001
s	2	0010
c	1	00110
x	1	00111
m	2	0100
o	2	0101
n	4	011
u	1	10000
l	1	10001
a	3	1001
r	1	10100
g	1	101010
p	1	101011
e	3	1011
i	3	1100
f	3	1101
	6	111

7. *0,8 бала* (Моделювання черги) Змоделюйте роботу сервісу з миття автомобілів відповідно до заданої діаграми.



Передбачається, що існує одна станція обслуговування. Кожен автомобіль вимагає рівно 10 хвилин на миття. У будь-який момент часу буде максимум 5 очікуючих машин у черзі на миття. Якщо автомобіль прибуває тоді, коли автомобіль миють, а 5 інших чекають у черзі, виводиться повідомлення ‘‘overflow’’, а новий автомобіль не враховується. Повідомлення про помилки мають друкуватись тоді, коли введений час прибуття не цілий, менше за 0 або більше за тривалість робочого дня (sentinel), або менше, ніж попередній час прибуття.

Середній час очікування визначається додаванням часу очікування кожного автомобіля та діленням суми на їх кількість. Докладніше щодо прибуттів та відправлень:

- Якщо прибуття та відправка відбуваються в одну і ту ж хвилину, першим обробляється відправка.
- Якщо автомобіль прибуває першим, його починають мити негайно і не ставлять у чергу.
- Автомобіль залишає чергу та завершує очікування, як тільки починається його 10-хвилинний цикл миття.

Приклад вводу може бути наступним:

5 5 7 12 12 13 14 18 19 25 999 (sentinel)

Для обчислення часу очікування кожної машини, слід віднімати її час прибуття від часу початку її миття. Перший час прибуття (t=5) не передбачає очікування, тому відповідний час дорівнює 0. Для другого автомобіля час прибуття такий же, тому автомобіль стає в чергу, а потім виходить з неї при початку свого миття (перша машина виїжджає з СТО в момент часу t = 15). Тому час очікування другого автомобіля - 10 хвилин. Повне моделювання наведено в таблиці:

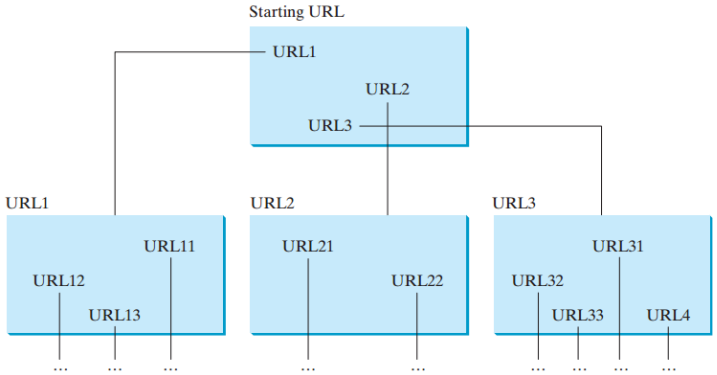
Arrival Time	Time Dequeued	Waiting Time
5		0
5	15	10
7	25	18
12	35	23
12	45	33
13	55	42
14	–	Overflow
18	65	47
19	–	Overflow
25	75	50

Сума часів очікування складає 223. Кількість автомобілів – 8 (переповнення в моменти t = 14 і t = 19 хвилин не враховуються), тому середній час очікування – 27.875 хвилин.

У класі CarWash передбачено наступні методи та конструктори:

- public CarWash() – ініціалізує об’єкт типу CarWash.
- public LinkedList<String> process (int nextArrivalTime) - обробляє всі події з поточного часу до заданого часу прибуття наступного автомобіля nextArrivalTime. Повертає історію миття автомобілів.
- public LinkedList<String> finishUp() – миє всі автомобілі, які залишились немитими після останнього прибуття. Повертає історію миття автомобілів після обробки всіх машин, що прибули.
- public LinkedList<String> getResults() – повертає історію прибуттів та відбуттів об’єкта типу CarWash і середній час очікування.

8. *0,7 бала* (Пошуковий робот) Змоделюйте роботу пошукового робота, який працює так, як зображено на рисунку.



Web-сторінка відкривається з трьома посиланнями: URL1, URL2, URL3.

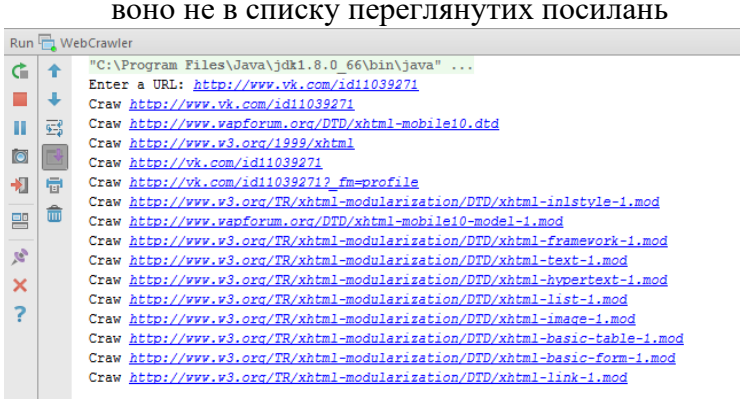
- Перехід по URL1 веде до сторінки, що містить URL11, URL12 та URL13.
- Перехід по URL2 веде до сторінки, яка містить URL21 та URL22.
- Перехід по URL3 веде до сторінки з URL31, URL32, URL33, URL34.

Продовжуйте обхід Web, переходячи за новими гіперпосиланнями. Процес може тривати безкінечно, якщо не введемо обмеження на кількість переглянутих сторінок, наприклад, до 100.

Програма слідує за посиланнями для обходу Web. Щоб переконатись, що кожен URL обходиться лише раз, програма підтримує два списки посилань: один містить посилання, які очікують обходу, а інший – які вже пройдено. Алгоритм роботи можна описати так:

Додати початковий URL у список очікуючих посилань;
Поки список очікуючих посилань непорожній, а розмір списку переглянутих посилань не перевищує 100
    Видалити посилання зі списку очікуючих посилань
    Якщо це посилання не в списку пройдених
        Додати в список пройдених посилань
        Вивести посилання
        Зчитати сторінку за цим посиланням і для кожного посилання на цій сторінці
            Додати знайдене посилання в список очікуючих посилань, якщо воно не в списку переглянутих посилань

Приклад виводу



Для роботи з посиланнями використовуйте клас `java.net.URL`.