Лабораторна робота №10

Тема: Застосування фреймворку Java Collections для роботи з колекціями.

Мета: Навчитися створювати та реалізовувати додатки з використанням колекцій у програмах на Java.

Обладнання: ОС Windows, інтегроване середовище JetBrains IntelliJ IDEA.

1. Теоретичні відомості.

До цього моменту часу ми здебільшого вивчали властивості та методи обробки поодиноких даних або їхніх найпростіших агрегатів - масивів та рядків. Ми вивчали основи моделювання окремих речей та явищ реального світу шляхом спрощення та формалізації наших уявлень про ці об'єкти. Але реальний світ вимагає від нас набагато більш комплексного погляду, адже часто сутність та взаємозв'язок речей та явищ неможливо або надто складно представити у вигляді простих агрегатів даних чи примітивних моделей. Тому, щоб дослідити взаємодію елементів реального світу не тільки в просторі, а й в часі, мови програмування повинні були набути відповідний інструментарій. І він існує, у вигляді бібліотек для колекціонування даних, у вигляді шаблонів проєктування тощо. Ці складніші абстракції та засоби ми почнемо вивчати з колекцій даних.

Колекції даних - це спосіб організації даних, призначений для зображення просторового поєднання реальних речей та явищ за певною ознакою або для моделювання зміни елементів реального світу у часі. Час та простір нерозривно пов'язані між собою. Тож, за умови правильного вибору типу колекції та алгоритмів над нею, існує можливість моделювати стан та поведінку об'єктів з її складу як у часі, так і в просторі.

У свідомості людини існує кілька типів колекцій, без знання властивостей та поведінки яких, було б неможливе існування людини у навколишньому світі. До умовно просторових колекцій можна віднести колекцію різних об'єктів, як найпоширеніший випадок, колекцію однакових об'єктів, як окремий випадок попередньої колекції, колекцію унікальних об'єктів, як окремий випадок все тієї ж колекції різних об'єктів, та, нарешті, індексовану колекцію, тобто таку, де є впорядкованість та порядковий номер елемента важить. Наприклад, з колекцією різних об'єктів, людина стикається змалку, спостерігаючи за речами у кімнаті. З колекцією однакових елементів ми також знайомимось у дитинстві, коли вчимося абстрагуватися, поєднуючи несхожі речі за спільними рисами. Наприклад, це відбувається під час використання дитиною конструктора. Колекції унікальних елементів більшість з нас також пізнає під час дитячих ігор, спостерігаючи за поєднанням в колекції чи групи елементів реального світу, які, ніби майже схожі, але все ж таки розрізняються й не повторюються. Нарешті, індексовану колекцію ми засвоюємо, коли вчимося впорядковувати простір чи ставати частиною впорядкованого простору. До умовно часових колекцій належать мережі у всіх своїх різновидах, Але ці абстракції набагато складніші, ніж умовно просторові

колекції. Тому їхніми властивостями ми вчимося користуватися протягом всього життя, часом успішно, а буває, що й ні.

В мові Java для роботи з колекціями існує спеціальний набір класів та функцій (в літературі такі інструменти називають "фреймворк"). Пакет java.util містить в собі елементи фреймворку Java Collections, в тому числі класи та інтерфейси.

Часто використання колекцій відбувається через використання інтерфейсів, а не безпосередньо класів. Це явище має природне пояснення. Коли ми створюємо будь-який складний об'єкт, існує кілька груп питань щодо нього. Зокрема, питання його внутрішньої будови та питання функціонала, заради якого він створюється. І часто буває так, що різні за внутрішньою будовою об'єкти виконують одні й ті самі функції. Таким чином, стає зрозуміло, що опис функціонала та внутрішньої будови варто розділити, що було показано в розділі, присвяченому інтерфейсам. Відповідно, для колекцій можна застосувати ті ж правила, оскільки колекція може враховувати особливості обчислювального пристрою, де її використовують, при цьому повністю зберігаючи функціональність, характерну для свого типу. Інтерфейси також можуть мати ієрархію. Наприклад, впорядкована множина є різновидом невпорядкованої множини тощо.

1.1 Елементи фреймворку Java Collections.

Фреймворк Java Collections містить ієрархію інтерфейсів та класів, зображену на рисунку 1.1 та описану в таблиці 1.1.

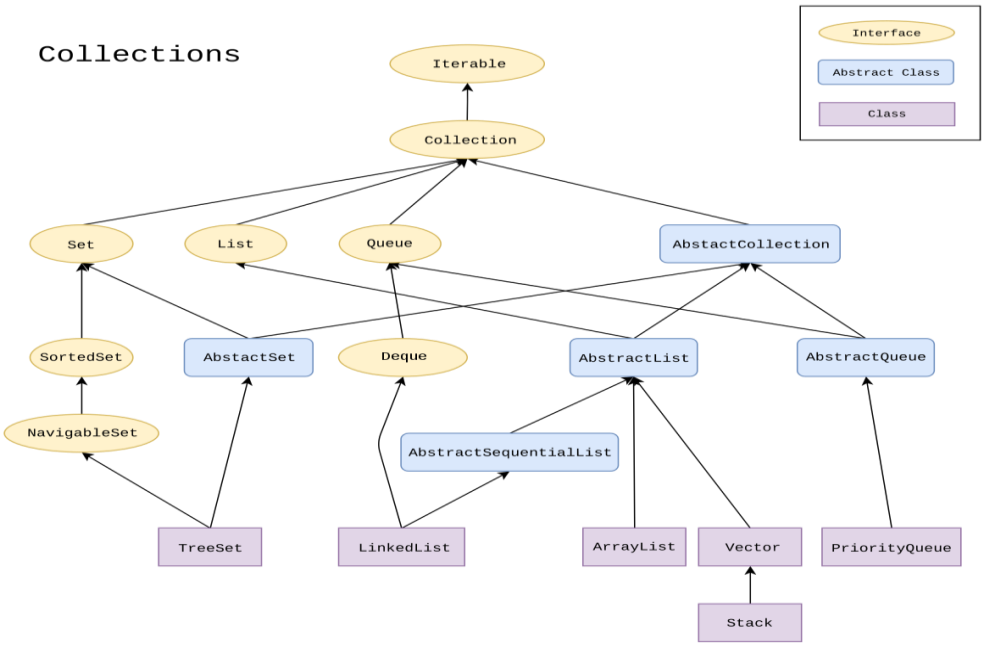

Рисунок 1.1. Ієрархія типів фреймворку Java Collections.

Таблиця 1.1. Інтерфейси та класи з Java Collections.

Інтерфейси та класи	**Опис**
Iterable	Ітератор є ітеративним класом. Основне завдання ітератора – видавати по одному елементу.
Collection	Інтерфейс Collection є узагальненим і розширює інтерфейс Iterable, тому всі об'єкти колекцій можна перебирати в циклі типу for-each.
Set	Це невпорядкована Collection (unordered Collection), що не допускає дублювання елементів і містить максимум 1 null-елемент. Якщо ви навмисно додасте дубльований елемент у Set, ця дія буде проігнорована і Set не зміниться.
List	Це interface у Java колекції інтерфейсів та класів (Java Collection Framework). Оскільки це під інтерфейс Collection, він має всі функції Collection. Крім того, він забезпечує основу для підтримки впорядкованої Collection (ordered Collection).
Queue	Інтерфейс Queue розширює Collection і оголошує поведінку черг, які є списком з дисципліною "перший увійшов, перший вийшов" (FIFO). Існують різні типи черг, у яких порядок ґрунтується на деякому критерії. Черги не можуть зберігати значення null.
AbstractCollection	Цей class забезпечує скелетну реалізацію Collection інтерфейсу, щоб мінімізувати зусилля, необхідне для реалізації цього інтерфейсу
SortedSet	Інтерфейс SortedSet призначений для створення колекцій, які зберігають елементи у відсортованому вигляді (сортування за зростанням). SortedSet розширює інтерфейс Set, тому така колекція знову ж таки зберігає тільки унікальні значення.
AbstractSet	Цей клас забезпечує скелетну реалізацію інтерфейсу Set, щоб мінімізувати зусилля, потрібне для реалізації цього інтерфейсу.

Інтерфейси та класи	**Опис**
Deque	Це підінтерфейс of Queue, що надає методи для вставки елемента в початок або кінець, а також методи доступу або видалення його першого або останнього елемента.
AbstractList	Цей клас надає скелетну реалізацію інтерфейсу List, щоб мінімізувати зусилля, необхідні реалізації цього інтерфейсу, підтримуваного сховищем даних «довільного доступу» (наприклад, масивом).
AbstractQueue	Цей class забезпечує скелетні реалізації деяких операцій Queue. Реалізації у цьому class є відповідними, коли основна реалізація не дозволяє елементи null.
NavigableSet	NavigableSet це під інтерфейс of SortedSet, тому він працює як SortedSet. Крім того, він має додаткові методи, які дозволяють переміщатися та знаходити елементи.
AbstractSequentialList	Цей клас забезпечує скелетну реалізацію інтерфейсу List, щоб мінімізувати зусилля, коли необхідно реалізовувати цей інтерфейс, підтриманий "з послідовним доступом" сховищем даних (наприклад, пов'язаний список). Для даних довільного доступу (таких як масив), AbstractList повинен використовуватися на перевагу до цього класу.
Treeset	Узагальнений клас TreeSet є структурою даних у вигляді дерева, в якому всі об'єкти зберігаються у відсортованому вигляді за зростанням. TreeSet є спадкоємцем класу AbstractSet і реалізує інтерфейс NavigableSet, отже, інтерфейс SortedSet.
LinkedList	Це клас, що реалізує два інтерфейси - List і Deque. Це забезпечує можливість створення двонапрямної черги з будь-яких (зокрема і null) елементів. Кожен об'єкт, поміщений у пов'язаний список, є вузлом (нодом).

Інтерфейси та класи	Опис
ArrayList	Це список, який дуже схожий на масив, за винятком того, що він має довільну довжину та цілу низку методів, які дозволяють додавати, видаляти, замінювати елементи списку в будь-який час і в будь-якому місці.
Vector	Клас Vector реалізує динамічний масив. Він схожий на ArrayList, але з двома відмінностями: Vector синхронізований. Vector містить багато застарілих методів, які є частиною структури колекцій.
PriorityQueue	Це необмежена (unbounded) черга. Елементи сортуються у порядку зростання, і допускається дублювання елементів.
Stack	У Java, Stack лише визначає стандартний конструктор, який створює порожній стек. Stack включає всі методи, визначені Vector, та самостійно додає кілька своїх власних.

1.2 Ітератори та інтерфейс Iterable.

Колекція, як структура даних, відрізняються від інших агрегатів даних тим, що кожен її елемент може бути доступний в результаті обходу колекції в циклі. Такою самою властивістю вирізняються й звичайні масиви, але колекція не схожа на масив, оскільки характер збереження її елементів в пам'яті обчислювального пристрою зовсім інший. Це питання було розглянуто в лабораторній роботі, присвяченій масивам. Таким чином, фреймворк Java Collections містить абстракцію набору даних який можна обійти в циклі у вигляді інтерфейсу Iterable, який є суперінтерфейсом для всіх інших абстракцій колекціонованих даних. В таблиці 1.2 описані методи цього інтерфейсу.

Таблиця 1.2. Методи інтерфейсу Iterable.

Метод	Опис
default void forEach(Consumer<? super T> action)	Застосовує подану дію для кожного елемента групи, що підтримує інтерфейс Iterable. Цей метод може збудити виключну ситуацію NullPointerException.

Метод	Опис
Iterator<T> iterator()	Повертає ітератор на елементи групи, що підтримує інтерфейс Iterable.
default Spliterator<T> spliterator()	Повертає Spliterator на елементи групи, що підтримує Iterable.

Як можна побачити з наведеної таблиці, окрім абстракції набору даних фреймворк містить ще й спеціальний клас, який можна використовувати у якості вказівника на певний елемент колекції в кожний окремий момент часу. Це нагадує лічильник циклу, за допомогою якого ми отримували доступ до елементів масиву. Такий уявний вказівник називають ітератором. Методи класу Iterator наведені у таблиці 1.3.

Таблиця 1.3. Методи класу Iterator.

Метод	Опис
default void forEachRemaining(Consumer<? super E> action)	Виконує подану дію для кожного елементу колекції, поки всі решта елементів не будуть оброблені або поки не виникне виняткова ситуація.
boolean hasNext()	Повертає true, якщо в колекції ще залишились необроблені елементи в колекції
E next()	Повертає наступний елемент колекції
default void remove()	Видалення поточного елемента з колекції

Використання ітераторів доволі схоже на використання лічильників циклу щодо масивів.

Розгляньмо такий приклад: нехай існує якась колекція (список, вектор, мапа чи будь-що інше), яка має ім'я elements, та складається з об'єктів класу Element. Тоді її обхід можна здійснити у два способи.

Спосіб 1.

```
for (Element obj: elements) {
        ...
    }
```

Починаючи з Java версії 8, в мові з'явився модифікований оператор for, який дозволяє використовувати ітератор за спрощеною схемою.

Спосіб 2.

```
for (Iterator it = elements.iterator(); it.hasNext();) {
        Object obj = it.next();
        ...
    }
```

Зауважте, що попри те, що цей вигляд циклу схожий на звичайний for, часто читання наступного елемента за допомогою метода next() виконується в тілі циклу, як показано вище.

Розгляньмо практичний приклад.

```
import java.util.*;

public class Test {
    public static void main(String[] argv) {
        // Оголошення звичайного масива
        Integer[] array = {1, 2, 3, 4, 5};
        // Виводимо елементи в рядок
        for (int i=0; i<array.length; i++) {
            System.out.print(array[i]);
            System.out.print("  ");
        }
        System.out.println();
        // створюємо індексовану колекцію, схожу на масив
        Vector<Integer> vector = new Vector<>(array.length);
        // додаємо в неї елементи зі створеного раніше масиву
        for (int i=0; i<array.length; i++) {
            vector.add(array[i]);
        }
        //  та  виводимо  її  вміст  за  допомогою  спрощеного
синтаксису
        for (Integer i: vector) {
            System.out.print(i);
            System.out.print("  ");
        }
        System.out.println();
        //  створення  інакшої  колекції  (списку)  з  елементів
масиву
        List<Integer> lst = Arrays.asList(array);
        //  вивід  елементів  масиву  за  допомогою  звичайного
ітератора
        for (Iterator it = lst.iterator(); it.hasNext(); ) {
            Object obj = it.next();
            System.out.print(obj);
```

```
                System.out.print("  ");
            }
            System.out.println();
        }
    }
```

Увага! Ітератор - це інструмент, за допомогою якого можна отримувати доступ до значень елементів колекції. Але спроба змінити відповідні значення в колекції зазнає невдачі. Це відбувається через те, що ітератор опрацьовує копії значень елементів колекції, з яких фактично утворюється потік. Про потоки мова піде у наступних роботах. Проте це не заважає використовувати ітератор вже зараз.

1.3 Інтерфейс Collection.

Цей інтерфейс є базовим до інших інтерфейсів колекцій, як показано на рисунку 1. Він описує абстрактний набір, який можна обійти циклом, безвідносно до того, чи є в ньому однакові або повторювані елементи, та чи можна до них звернутись за індексом.

Оголошення інтерфейсу Collection виглядає так:

```
public interface Collection<E> extends Iterable<E> {
    ...
}
```

Основні методи інтерфейсу Collection наведені в таблиці 1.4.

Таблиця 1.4. Методи інтерфейсу Collection.

Метод	**Опис**
boolean add(E e)	Додає елемент в колекцію
boolean addAll(Collection<? extends E> c)	Додає елементи з поданої колекції до поточної.
void clear()	Видаляє всі елементи з поточної колекції
boolean contains(Object o)	Повертає true, якщо поданий елемент міститься в поточній колекції
boolean containsAll(Collection<?> c)	Повертає true, якщо в поточній колекції містяться усі елементи з поданої колекції.

Метод	Опис
boolean equals(Object o)	Повертає true, якщо поданий об'єкт дорівнює поточній колекції.
int hashCode()	Повертає хеш-код поточної колекції.
boolean isEmpty()	Повертає true, якщо поточна колекція порожня.
Iterator<E> iterator()	Повертає ітератор на для поточної колекції
default Stream<E> parallelStream()	Повертає паралельний потік для поточної колекції.
boolean remove(Object o)	Видаляє поданий елемент з поточної колекції
boolean removeAll(Collection<?> c)	Видаляє всі елементи з поданої колекції з поточної колекції
default boolean removeIf(Predicate<? super E> filter)	Видаляє всі елементи з поточної колекції, які задовільняють поданій умові
boolean retainAll(Collection<?> c)	Залишає в поточній колекції тільки ті елементи, які містяться у поданій колекції
int size()	Повертає кількість елементів у поточній колекції
default Spliterator<E> spliterator()	Повертає Spliterator для поточної колекції
default Stream<E> stream()	Повертає послідовний потік для поточної колекції.
Object[] toArray()	Повертає масив, що містить усі елементи поточної колекції
<T> T[] toArray(T[] a)	Повертає масив заданого типу з усіх елементів, що містяться в поточній колекції

Як видно з наведеного опису, найзагальнішими рисами будь-якої колекції є додавання та видалення елементів, перевірка на наявність елементів, отримання поточного розміру колекції, а також отримання ітератора на колекцію та потоку елементів. Тобто всі ці операції не накладають жодних обмежень на характер колекції або тип її елементів. Використання колекцій можливо, як використання будь-якого іншого інтерфейсу - завдяки механізму поліморфізму. Розгляньмо приклад:

```
import java.util.*;

public class Test1 {
    public static void main(String[] argv) {
        /*  Оголошуємо вектор (індексовану колекцію) та
обробляємо її за допомогою суперінтерфейсу Collection */
         Collection<Integer> c = new Vector<Integer>(5);
         for (int i=0; i<5; i++) {
             c.add(i+1);
         }
         // намагаємось змінити елементи колекції, але марно.
         // причина цього була описана у попередньому розділі.
         // проте код компілюється без помилок
         for (Integer i: c) {
             i = 0;
         }
         // друкуємо вміст вектора на екран
         for (Iterator it = c.iterator(); it.hasNext();) {
             Object obj = it.next();
             System.out.println(obj);
         }

        // ще один спосіб виведення
        c.forEach((x) -> {
             System.out.println(x);
         });
    }
}
```

1.4 Клас Vector.

Клас вектор є індексованою колекцією, за сутністю схожою з масивом. Елементи вектора так само можуть бути доступні та змінені за індексом, як це відбувається з елементами масиву. Зручність використання вектора замість масиву полягає у можливості застосування до векторів шаблонних алгоритмів, таких як алгоритми пошуку та сортування, без синтаксичних складнощів. Методи цього класу наведені у таблиці 1.5.

Таблиця 1.5. Методи класи Vector.

Метод	**Опис**
boolean add(E e)	Додає елементи в кінець вектора
void add(int index, E element)	Вставляє елемент в позицію за вказаним індексом
boolean addAll(int index, Collection<? extends E> c)	Вставляє елементи з поданої колекції у зазначену позицію вектора
void addElement(E obj)	Додає елемент в кінець вектора
int capacity()	Повертає поточну місткість вектора. Під час створення вектора, зазвичай зазначається кількість елементів, яка має в ньому бути. Ця кількість, на відміну від масиву, не є остаточною, і може гнучко змінюватись, але в будь-який час відомо, скільки пам'яті зарезервовано для розміщення елементів вектора.
void clear()	Видаляє всі елементи з вектора
Object clone()	Повертає копію масиву
void copyInto(Object[] anArray)	Копіює елементи вектора у поданий масив
E elementAt(int index)	Повертає елемент вектора із зазначеним індексом
void ensureCapacity(int minCapacity)	За потреби, збільшує місткість вектору
boolean equals(Object o)	Порівнює поданий об'єкт та поточний вектор та з'ясовує, чи вони рівні. Якщо так, повертає true, інакше - false.
E firstElement()	Повертає перший елемент вектора
E get(int index)	Повертає елемент за поданим індексом
int hashCode()	Повертає хеш-код вектора

Метод	Опис
int indexOf(Object o)	Повертає індекс поданого елементу вектора
int indexOf(Object o, int index)	Повертає індекс першої, після вказаного індексу, появи у векторі поданого елементу
void insertElementAt(E obj, int index)	Вставляє поданий елемент у вектор за вказаним індексом
boolean isEmpty()	Визначає, чи сектор порожній, чи ні. Повертає true, якщо порожній, та false в іншому випадаку.
E lastElement()	Повертає останній елемент з вектора
int lastIndexOf(Object o)	Повертає останній індекс появи елемента у векторі
int lastIndexOf(Object o, int index)	Повертає останній, перед вказаним індексом, індекс поданого елементу у векторі
E remove(int index)	Видаляє з вектора елемент за поданим індексом.
boolean remove(Object o)	Видаляє поданий елемент з вектора
void removeAllElements()	Видаляє всі елементи вектора
boolean removeElement(Object obj)	Видаляє поданий елемент з вектора
void removeElementAt(int index)	Видаляє з вектора елемент за поданим індексом
boolean retainAll(Collection<?> c)	Повертає лише ті елементи вектора, які також містяться у поданій колекції
E set(int index, E element)	Заміняє значення елемента з вказаним індексом на подане
void setElementAt(E obj, int index)	Замінює елемент вектора за вказаним індексом на поданий
void setSize(int newSize)	Встановлює розмір вектора

Метод	Опис
int size()	Повертає поточний розмір вектора
void sort(Comparator<? super E> c)	Сортує вектор в порядку, створеному поданим генератором
List<E> subList(int fromIndex, int toIndex)	Повертає список з елементів вектора, розташованих між початковим та кінцевим індексами
Object[] toArray()	повертає масив об'єктів з, створений з елементів вектора
String toString()	повертає вектор у вигляді рядка символів
void trimToSize()	зменшує місткість вектора до вказаного обсягу

2. Порядок виконання роботи

2.1 Для здобуття мінімально необхідного похідного балу за виконання лабораторної роботи з таблиці 2.1 обрати завдання до виконання. Номер варіанту за списком для кожної підгрупи.

2.3 Для здобуття більш високого балу за виконання лабораторної роботи виконати додатково одне завдання (рівень 1) або два завдання (рівень 2) з таблиці 2.2.

2.3 Створити та надати викладачеві звіт з виконання лабораторної роботи. Звіт повинен містити тему, мету та обладнання для виконання лабораторної роботи, короткий опис постанови задачи для кожного з завдань, лістинг програми, скріншоти результатів тестування програми.

3. Варіанти індивідуальних завдань

Таблиця 2.1

Варіант	Завдання
1	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими цілими числами в діапазоні від -100 до 100 таким чином, щоб кінцевий розмір колекції перевищував початкову ємність. Обійти колекцію за допомогою ітератора, замінивши від'ємні елементи на нульове значення. Результати вивести на екран.

Варіант	Завдання
2	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими дійсними числами у діапазоні від -15.4 до -10.8 таким чином, щоб кінцевий розмір колекції перевищував початкову ємність. Обійти колекцію за допомогою ітератора, виконавши обчислення значення модуля для кожного з елементів вектора. Записати результат обчислення замість відповідного елемента вектора. Результати вивести на екран.
3	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими цілими числами в діапазоні від -200 до 200 таким чином, щоб кінцевий розмір колекції перевищував початкову ємність. Обійти колекцію за допомогою ітератора, виконавши обчислення відхилення кожного елемента від середнього значення в початковому векторі, та записати їх у вектор замість відповідних елементів. Результати вивести на екран.
4	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими дійсними числами у діапазоні від -16.1 до 8.8. Обійти колекцію за допомогою ітератора, виконавши округлення до найближчого більшого цілого для кожного з елементів вектора, записавши результат округлення замість відповідного елемента. Результати вивести на екран.
5	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими цілими числами в діапазоні від-150 до 150 таким чином, щоб кінцевий розмір колекції перевищував початкову ємність. Обійти колекцію за допомогою ітератора, виконавши інверсію знаку кожного додатнього елемента колекції та замінити на нульове значення інші елементи колекції. Результати вивести на екран.
6	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими дійсними числами в діапазоні від -10.5 до 7.8 таким чином, щоб кінцевий розмір колекції перевищував початкову ємність. Обійти колекцію за допомогою ітератора, виконавши округлення до найближчого меншого цілого для кожного з елементів вектора,

Варіант	Завдання
	записавши результат округлення замість відповідного елемента. Результати вивести на екран.
7	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими цілими числами в діапазоні від-120 до 120 таким чином, щоб кінцевий розмір колекції перевищував початкову ємність. Обійти колекцію за допомогою ітератора, замінивши всі додатні елементи на максимальне значення елементів колекції, інші елементи видалити. Вивести на екран початковий та змінений вектор.
8	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими дійсними числами у діапазоні від -9.6 до 5.2 таким чином, щоб кінцевий розмір колекції перевищував початкову ємність. Обійти колекцію за допомогою ітератора, виконавши обчислення корня квадратного із додатніх елементів та замінити на нульове значення інші елементи колекції. Результати вивести на екран.
9	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими цілими числами у діапазоні від -50 до 50 таким чином, щоб кінцевий розмір колекції перевищував початкову ємність. Обійти колекцію за допомогою ітератора, замінивши всі від'ємні елементи на мінімальне значення елементів колекції, інші елементи не змінювати. Результати вивести на екран.
10	Створити колекцію Vector, задавши за допомогою конструктора початкову ємність. Заповнити колекцію випадковими дійсними числами у діапазоні від -5.9 до 9.2 таким чином, щоб кінцевий розмір колекції перевищував початкову ємність. Обійти колекцію за допомогою ітератора, видаливши додатні елементи вектора та замінивши від'ємні елементи на нульове значення. Вивести на екран початковий та змінений вектор.

Таблиця 2.2

Варіант	Завдання
1	Створити множину, заповнити її значеннями чисел, які користувач вводить з клавіатури у вигляді одного рядку, наприклад "1, 2, 3, 4, 4, 5, 7, 7, 7, 10,11, 11". Використовуючи інтерфейси Iterator

Варіант	Завдання
	та Set, обійти колекцію, видаляючи елементи в рядку, які повторюються. Вивести на екран змінений рядок.
2	Створити клас Book з полями name, cost, author. З об'єктів Book створити колекцію HashMap з парами значень - ім'я книги та об'єкт книги. Створити методи, які будуть виконувати наступні дії: • вивести весь набір книжок в колекції; • вивести набір книжок одного автора; • вивести набір книжок, вартість яких не перевищує задану.
3	Створити класс Student з полями ім'я, група, курс, середній бал. Створити колекцію з об'єктів Student. Створити методи, які будуть виконувати наступні дії: • вивести список студентів, які навчаються на заданому курсі; • видалити студентів, середній бал яких <3; • перевести на наступний курс студентів, середній бал яких >=3.
4	Є набір ємностей *Box(double width, double height, double depth)* для перевезення різних вантажів. Успадкований клас *HeavyBox(double weight)* має характеристику вантажопідйомності. Створити клас Вантажівка з характеристиками об'єму кузова та максимального навантаження машини. Створити колекцію Вантажівок з номером машини та ім'ям водія. Створити динамічний масив з об'єктів *HeavyBox*. Імітувати завантаження кожної машини перебираючи елементи динамічного масиву, розраховуючи скільки *HeavyBox* може поміститися в кожній з вантажівок. Результати вивести на екран.

4. Питання для самоконтролю

1. Охарактерізуйте структури даних, з яких може створюватися колекція.
2. Чому використання колекцій зручніше робити з використанням інтерфейсів, а не класів.
3. Визначте структури даних, при роботі з якими зручно використовувати такі інтерфейси як Set, List, Queue.
4. Який спеціальний клас можна використовувати у якості вказівника на певний елемент колекції в кожний окремий момент часу.
5. Які різновиди циклу for існують для обходу колекцій.