**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені ІГОРЯ СІКОРСЬКОГО”**

**ЗВІТ**

**з лабораторної роботи №5**

**з навчальної дисципліни “****Рекурсивні алгоритми”**

**Тема:**

**ЛІНІЙНІ ОДНОЗВ’ЯЗНІ ТА ДВОЗВ’ЯЗНІ СПИСКИ.**

**Виконав студент групи ТР–12**

Каркушевський Владислав

Лабораторну роботу захищено

з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Перевірив доцент кафедри**

\_\_\_\_\_\_\_\_\_\_\_ Андрій ОНИСЬКО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 року

**Київ 2022**

**Мета:** Метою лабораторної роботи є ознайомитися з основами роботи з двозв’язним списком, однозв’язним списком, стеком та чергою.

**Теоретична частина:**

Динамічні структури даних - це структури даних, пам'ять під які виділяється і звільняється в міру необхідності.

Динамічні струкури даних в процесі існування в пам'яті можуть змінювати не тільки число складових їх елементів, а й характер зв'язків між елементами.

При цьому не враховується зміна вмісту самих елементів даних. Така особливість динамічних структур, як зміна їх розміру і характеру відносин між елементами, призводить до того, що на етапі створення машинного коду програма-компілятор не може виділити для всієї структури в цілому ділянку пам'яті фіксованого розміру, а також не може зіставити з окремими компонентами структури конкретні адреси.

**Однозв’язаний список**

Однозв’язаний список — це лінійна структура даних, в якій кожен елемент списку містить покажчик, який вказує на наступний елемент у списку. Кожен елемент у однозв’язаному списку називається вузлом. Кожен вузол має два компоненти: дані та вказівник поруч, який вказує на наступний вузол у списку. Перший вузол списку називається головним, а останній вузол списку називається хвостом. Останній вузол списку містить покажчик на значення null. До кожного вузла в списку можна отримати лінійний доступ, проходячи по списку від голови до хвоста.

**Двозв’язаний список**

Двозв’язаний список — це складний тип зв’язаного списку, в якому вузол містить вказівник як на попередній, так і на наступний вузол у послідовності. Отже, у двозв’язаному списку вузол складається з трьох частин: даних вузла, покажчика на наступний вузол у послідовності (наступний покажчик), покажчика на попередній вузол (попередній покажчик).

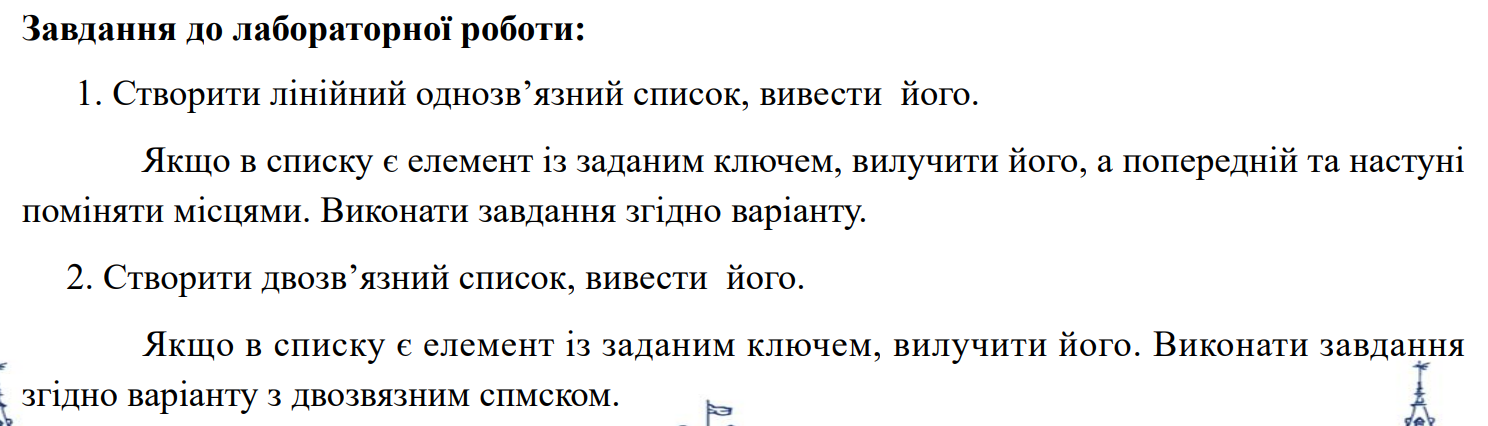
**Стек**

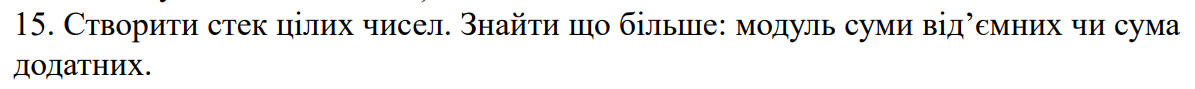
Стек — це лінійна структура даних, яка **використовується** для зберігання колекції об’єктів. Він заснований на принципі « **Останній прийшов першим вийшов** » (LIFO).

**Черга**

Черга — це інший вид лінійної структури даних, яка використовується для зберігання елементів, як і будь-яка інша структура даних, але особливим чином. Простими словами можна сказати, що черга — це тип структури даних у мові програмування Java, в якій зберігаються однотипні елементи. Компоненти в черзі зберігаються в режимі FIFO (першим прийшов, першим вийшов)

**Завдання**

****

****

**Код програми:**

**SingleLinkedList**

*import* java.util.*Iterator*;  
*import* java.util.NoSuchElementException;  
  
*public class* SingleLinkedList<E> *implements Iterable*<E> {  
   
 *private* Node<E> firstNode;  
 *private int* size = 0;  
  
 *public* SingleLinkedList() {*//конструктор* firstNode = *new* Node<E>(*null*, *null*);  
 }

*public void* clear() {*//метод для очищення списку* size = 0;  
 firstNode = *new* Node<E>(*null*, *null*);  
  
}

*public void* addFirst(E e) {*//метод для додавання елементу до списку у початок* Node<E> next = firstNode;  
 next.setElem(e);  
 firstNode = *new* Node<>(*null*, next);  
  
 size++;  
 }  
   
 *public void* removeFirst() {*//метод для видалення першого елементу списку* firstNode.setElem(*null*);  
 firstNode = getNext(firstNode);  
 size--;  
 }  
  
  
 *public void* remove(*int* index) {*//метод для видалення елементу списку за індексом  
 if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 *if* (index == 0) {  
 removeFirst();  
  
 } *else* {  
  
 Node<E> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < index - 1; i++) {  
 temp = temp.next;  
 }  
 Node<E> temp2 = temp.next;  
 temp.setNext(temp2.next);  
 temp2.setElem(*null*);  
 size--;  
  
 }  
   
 }  
   
 *public* E get(*int* index) {*//метод для знаходження елемента списку за заданим індексом  
 if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 Node<E> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < index; i++) {  
 temp = getNext(temp);  
 }  
  
 *return* temp.getElem();  
 }  
  
 *public void* set(*int* index, E elem) {*//метод для заміни елемента листа за заданим індексом  
 if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 Node<E> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < index; i++) {  
 temp = getNext(temp);  
 }  
 temp.setElem(elem);  
  
 }  
   
 *public int* size() {*//метод для виведення розміру  
 return* size;  
 }  
   
 *public boolean* isEmpty() {*//метод для перевірки листа на наявність елементів  
 return* size == 0;  
 }  
   
   
 @Override  
 *public* String toString() {*//Перевизначимо метод toString для виведення списку  
  
 if* (size == 0) *return* "List is clear";  
 StringBuilder sb = *new* StringBuilder();  
 sb.append("[");  
  
 Node<E> current = firstNode.getNext();  
 *while* (current != *null*) {  
  
 *if* (current.getElem() == *null*) {  
 current = getNext(current);  
 *continue*;  
 }  
  
 sb.append(current.getElem());  
 current = current.next;  
 *if* (current != *null*) sb.append(", ");  
 }  
  
 sb.append("]");  
 *return* sb.toString();  
 }  
   
 *private* Node<E> getNext(Node<E> current) {  
 *return* current.getNext();  
 }  
  
 @Override  
 *public Iterator* iterator() {*//визначимо ітератор  
 return new* Iterator() {  
 *int* counter = 0;  
  
 @Override  
 *public boolean* hasNext() {  
 *return* counter < size;  
 }  
  
 @Override  
 *public* Object next() {  
 *return* get(counter++);  
 }  
  
 @Override  
 *public void* remove() {  
 *throw new* UnsupportedOperationException("remove");  
 }  
  
 };  
 }  
  
 *private class* Node<E> {*//створимо клас Node  
  
  
 private* E elem;  
 *private* Node<E> next;  
  
  
 *private* Node(E elem, Node<E> next) {*//конструктор  
 this*.elem = elem;  
 *this*.next = next;  
 }  
  
 *public* E getElem() {  
 *return* elem;  
 }*//гетери і сетери  
  
 public void* setElem(E elem) {  
 *this*.elem = elem;  
 }  
  
 *public* Node<E> getNext() {  
 *return* next;  
 }  
  
 *public void* setNext(Node<E> next) {  
 *this*.next = next;  
 }  
  
 }  
}

**DoublyLinkedList**

*import* java.util.*Iterator*;  
*import* java.util.NoSuchElementException;  
  
  
*public class* DoublyLinkedList<E> *implements Iterable*<E> {  
  
  
 *private* Node<E> firstNode;  
 *private* Node<E> lastNode;  
 *private int* size = 0;  
  
  
 *public* DoublyLinkedList() {*//конструктор* lastNode = *new* Node<E>(*null*, firstNode, *null*);  
 firstNode = *new* Node<E>(*null*, *null*, lastNode);  
 }  
  
  
 *public int* size() {  
 *return* size;  
 }*//метод для виведення розміру  
  
 public boolean* isEmpty() {*//метод для перевірки листа на наявність елементів  
 return* size == 0;  
 }  
  
*public void* addFirst(E e) {*//метод для додавання елементу до списку у початок* Node<E> next = firstNode;  
 next.setElem(e);  
 firstNode = *new* Node<>(*null*, *null*, next);  
 *if* (isEmpty()) lastNode.setPrev(next);  
  
 next.setPrev(firstNode);  
 size++;  
}  
  
  
*public void* addLast(E e) {*//метод для додавання елементу до списку у кінець* Node<E> prev = lastNode;  
 prev.setElem(e);  
 lastNode = *new* Node<E>(*null*, prev, *null*);  
 *if* (isEmpty()) firstNode.setNext(prev);  
 prev.setNext(lastNode);  
 size++;  
}

*public void* removeFirst() {*//метод для видалення першого елементу списку  
 if* (isEmpty()) *throw new* NoSuchElementException("List is clear");  
 firstNode = firstNode.getNext();  
 size--;  
  
 }  
  
 *public void* removeLast() {*//метод для видалення останнього елементу списку  
 if* (isEmpty()) *throw new* NoSuchElementException("List is clear");  
 lastNode = lastNode.getPrev();  
 size--;  
 }  
  
 *public void* remove(*int* index) {*//метод для видалення елементу списку за індексом  
 if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 *if* (index == 0) {  
 removeFirst();  
  
 } *else if* (index == size() - 1) {  
 removeLast();  
  
 } *else* {  
  
 Node<E> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < index - 1; i++) {  
 temp = temp.next;  
 }  
 Node<E> temp2 = temp.next;  
 temp.setNext(temp2.next);  
 temp2.next.setPrev(temp);  
 temp2.setElem(*null*);  
 size--;  
  
 }  
  
 }  
  
 *public void* clear() {*//метод для очищення списку* size = 0;  
 lastNode = *new* Node<E>(*null*, firstNode, *null*);  
 firstNode = *new* Node<E>(*null*, *null*, lastNode);  
  
 }  
  
 *public* E get(*int* index) {*//метод для знаходження елемента списку за заданим індексом  
  
 if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 Node<E> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < index; i++) {  
 temp = getNext(temp);  
 }  
 *return* temp.getElem();  
 }  
  
 *public void* set(*int* index, E value) {*//метод для заміни елемента листа за заданим індексом  
 if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 Node<E> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < index; i++) {  
 temp = getNext(temp);  
 }  
 temp.setElem(value);  
 }  
  
  
 *public int* indexOf(E elem) {*//пошук індекса елементу списку* Node<E> temp = firstNode.getNext();  
 *int* index = -1;  
 *for* (*int* i = 0; i < size; i++) {  
 *if* (elem.toString().equals(temp.getElem().toString())) {  
 index = i;  
 *break*;  
 }  
 temp = getNext(temp);  
 }  
  
 *if* (index == -1) *throw new* NoSuchElementException("No element index");  
 *return* index;  
 }  
  
  
 @Override  
 *public* String toString() {*//Перевизначимо метод toString  
  
 if* (size == 0) *return* "List is clear";  
 StringBuilder sb = *new* StringBuilder();  
 sb.append("[");  
  
 Node<E> current = firstNode.getNext();  
  
 *while* (current != *null*) {  
  
 sb.append(current.getElem());  
 *if* (current.next == lastNode) {  
 *break*;  
 }  
 current = current.next;  
 *if* (current != *null*) sb.append(", ");  
 }  
  
 sb.append("]");  
 *return* sb.toString();  
 }  
  
  
 @Override  
 *public Iterator*<E> iterator() {*//визначимо ітератор  
 return new* Iterator<E>() {  
 *int* counter = 0;  
  
 @Override  
 *public boolean* hasNext() {  
 *return* counter < size;  
 }  
  
 @Override  
 *public* E next() {  
 *return* get(counter++);  
 }  
 };  
 }  
  
  
 *private* Node<E> getNext(Node<E> current) {  
 *return* current.getNext();  
 }  
  
 *private* Node<E> getPrev(Node<E> current) {  
 *return* current.getPrev();  
 }  
  
 *private class* Node<E> {*//створимо клас Node  
  
  
 private* E elem;  
 *private* Node<E> prev;  
 *private* Node<E> next;  
  
  
 *private* Node(E elem, Node<E> prev, Node<E> next) {*//конструктор  
 this*.elem = elem;  
 *this*.prev = prev;  
 *this*.next = next;  
 }  
  
 *public* E getElem() {  
 *return* elem;  
 }*//гетери і сетери  
  
 public void* setElem(E elem) {  
 *this*.elem = elem;  
 }  
  
 *public* Node<E> getNext() {  
 *return* next;  
 }  
  
 *public void* setNext(Node<E> next) {  
 *this*.next = next;  
 }  
  
 *public* Node<E> getPrev() {  
 *return* prev;  
 }  
  
 *public void* setPrev(Node<E> prev) {  
 *this*.prev = prev;  
 }  
  
  
 }  
}

**Stack**

*import* java.util.*Iterator*;  
*import* java.util.NoSuchElementException;  
  
*class* Stack<E> *implements Iterable*<E> {  
  
  
 *private* Node<E> firstNode;  
 *private int* size = 0;  
  
 *public* Stack() {  
 firstNode = *new* Node<E>(*null*, *null*);*//конструктор* }  
  
 *public int* size() {*//метод для виведення розміру стеку  
 return* size;  
 }*//метод для виведення розміру  
  
 public boolean* isEmpty() {  
 *return* size == 0;  
 }*//Перевіряє, чи стек є порожнім.  
  
  
 public* E pop() {*//Повертає елемент, що знаходиться у верхній частині стека, видаляючи його у процесі  
 if* (isEmpty()) *throw new* NoSuchElementException("Stack is clear");  
 E removeElem = firstNode.getNext().getElem();  
 firstNode.setElem(*null*);  
 firstNode = getNext(firstNode);  
 size--;  
 *return* removeElem;  
 }  
  
  
 *public* E peek() {*//Повертає елемент, що знаходиться у верхній частині стека, але не видаляє його.  
 if* (isEmpty()) *throw new* NoSuchElementException("Stack is clear");  
 *return* firstNode.getNext().getElem();  
 }  
  
  
 *public* E push(E elem) {*//додає елемент у стек* Node<E> next = firstNode;  
 next.setElem(elem);  
 firstNode = *new* Node<>(*null*, next);  
 size++;  
 *return* elem;  
 }  
  
 *public void* remove(*int* index) {*//видаляє елемент за індексом  
  
 if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 *if* (index == 0) {  
 pop();  
  
 } *else* {  
  
 Node<E> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < index - 1; i++) {  
 temp = temp.next;  
 }  
 Node<E> temp2 = temp.next;  
 temp.setNext(temp2.next);  
 temp2.setElem(*null*);  
 size--;  
  
 }  
  
 }  
  
  
 *public* E get(*int* index) {*//повертає елемент за індексом  
 if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 Node<E> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < index; i++) {  
 temp = getNext(temp);  
 }  
  
 *return* temp.getElem();  
 }  
  
  
 *public int* search(E element) {*//повертає індекс елемента у стеку  
 int* index = -1;  
 Node<E> temp = firstNode.getNext();  
 *for* (*int* i = 0; i < size(); i++) {  
 *if* (temp.getElem() == element) {  
 System.***out***.println(i);  
 index = i;  
 *break*;  
 }  
 temp = getNext(temp);  
 }  
 *return* index;  
 }  
  
 @Override  
 *public* String toString() {*//Перевизначимо метод toString  
  
 if* (size == 0) *return* "Stack is clear";  
 StringBuilder sb = *new* StringBuilder();  
 sb.append("[");  
  
 Node<E> current = firstNode.getNext();  
 *while* (current != *null*) {  
 sb.append(current.getElem());  
 current = current.next;  
 *if* (current != *null*) sb.append(", ");  
 }  
 sb.append("]");  
 *return* sb.toString();  
 }  
  
 @Override  
 *public Iterator* iterator() {*//створимо ітератор  
 return new* Iterator() {  
 *int* counter = 0;  
  
 @Override  
 *public boolean* hasNext() {  
 *return* counter < size;  
 }  
  
 @Override  
 *public* Object next() {  
 *return* get(counter++);  
 }  
 };  
 }  
  
 *private class* Node<E> {*//створимо клас Node  
  
 private* E elem;  
 *private* Node<E> next;  
  
  
 *private* Node(E elem, Node<E> next) {*//створимо конструктор  
 this*.elem = elem;  
 *this*.next = next;  
 }  
  
 *public* E getElem() {*//створимо сетери і гетери  
 return* elem;  
 }*//створимо гетери і сетери  
  
 public void* setElem(E elem) {  
 *this*.elem = elem;  
 }  
  
 *public* Node<E> getNext() {  
 *return* next;  
 }  
  
 *public void* setNext(Node<E> next) {  
 *this*.next = next;  
 }  
 }  
  
 *private* Node<E> getNext(Node<E> current) {  
 *return* current.getNext();  
 }  
  
}

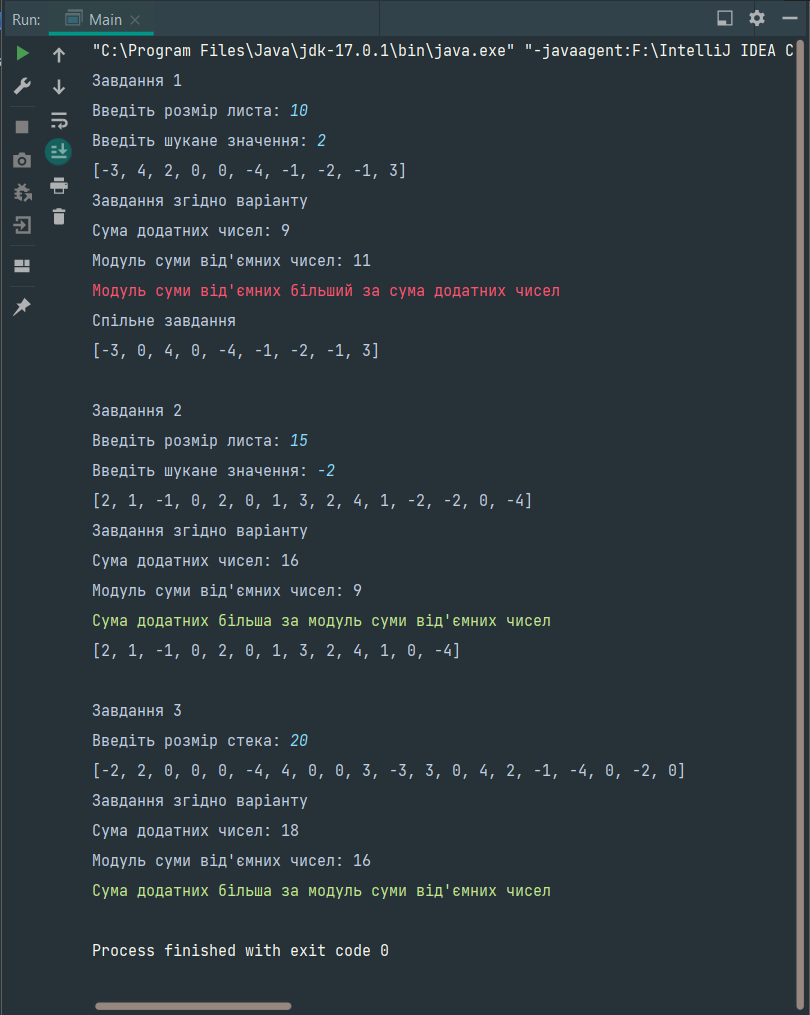
**Queue**

*import* java.util.*Iterator*;  
*import* java.util.NoSuchElementException;  
  
*public class* Queue<E> *implements Iterable*<E> {  
  
  
 *private* Node<E> firstNode;  
 *private* Node<E> lastNode;  
 *private int* size;  
  
 *public* Queue() {*//конструктор* lastNode = *new* Node<>(*null*, *null*);  
 firstNode = *new* Node<>(*null*, *null*);  
 }  
  
 *public boolean* isEmpty() {*//метод для перевірки листа на наявність елементів  
 return* size == 0;  
 }  
  
 *public int* size() {*//метод для виведення розміру  
 return* size;  
 }  
  
 *public void* enqueue(E item) {*//метод для додавання елементу до списку* Node<E> oldlast = lastNode;  
 lastNode = *new* Node(item, *null*);  
 *if* (isEmpty()) firstNode = lastNode;  
 *else* oldlast.next = lastNode;  
 size++;  
  
 }  
  
 *public* E dequeue() {*//метод для видалення першого елементу списку  
 if* (isEmpty()) *throw new* NoSuchElementException("Queue is clear");  
 E item = firstNode.getElem();  
 firstNode = firstNode.next;  
 size--;  
 *if* (isEmpty()) lastNode = *null*;  
 *return* item;  
 }  
  
 *public void* remove(*int* index) {  
 *if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 *if* (index == 0) {  
 dequeue();  
  
 } *else* {  
  
 Node<E> temp = firstNode;  
 *for* (*int* i = 0; i < index - 1; i++) {  
 temp = temp.next;  
 }  
 Node<E> temp2 = temp.next;  
 temp.setNext(temp2.next);  
 temp2.setElem(*null*);  
 size--;  
  
 }  
  
 }  
  
  
 *public* E head() {*//повертаємо голову черги  
 if* (isEmpty()) *throw new* NoSuchElementException("Queue is clear");  
 *return* firstNode.getElem();  
 }  
  
 *public* E tail() {*//повертаємо хвіст черги  
 if* (isEmpty()) *throw new* NoSuchElementException("Queue is clear");  
 *return* lastNode.getElem();  
 }  
  
 *public* E get(*int* index) {*//метод для знаходження елемента списку за заданим індексом  
  
 if* (index < 0 || index >= size) *throw new* NoSuchElementException("No element index");  
 Node<E> temp = firstNode;  
 *for* (*int* i = 0; i < index; i++) {  
 temp = getNext(temp);  
 }  
 *return* temp.getElem();  
 }  
  
  
 @Override  
 *public* String toString() {*//Перевизначимо метод toString  
  
 if* (size == 0) *return* "Queue is clear";  
 StringBuilder sb = *new* StringBuilder();  
 sb.append("[");  
  
 Node<E> current = firstNode;  
  
 *while* (current != *null*) {  
 sb.append(current.getElem());  
 current = current.next;  
 *if* (current != *null*) sb.append(", ");  
 }  
  
 sb.append("]");  
 *return* sb.toString();  
 }  
  
 @Override  
 *public Iterator*<E> iterator() {*//визначимо ітератор  
 return new* Iterator<E>() {  
 *int* counter = 0;  
  
 @Override  
 *public boolean* hasNext() {  
 *return* counter < size;  
 }  
  
 @Override  
 *public* E next() {  
 *return* get(counter++);  
 }  
 };  
 }  
  
 *private* Node<E> getNext(Node<E> current) {  
 *return* current.getNext();  
 }  
  
 *private class* Node<E> {*//створимо клас Node  
  
  
 private* E elem;  
 *private* Node<E> next;  
  
  
 *private* Node(E elem, Node<E> next) {  
 *this*.elem = elem;  
  
 *this*.next = next;  
 }  
  
 *public* E getElem() {  
 *return* elem;  
 }  
  
 *public void* setElem(E elem) {  
 *this*.elem = elem;  
 }  
  
 *public* Node<E> getNext() {  
 *return* next;  
 }  
  
 *public void* setNext(Node<E> next) {  
 *this*.next = next;  
 }  
  
  
 }  
  
}

**Main**

*import* java.util.Scanner;*//імпортуємо сканер  
  
public class* Main {  
 *public static void* main(String[] args) {  
  
 Scanner sc = *new* Scanner(System.***in***);*//викликаємо сканер* System.***out***.println("Завдання 1 ");  
 SingleLinkedList<Integer> sList = *new* SingleLinkedList<>();  
  
 System.***out***.print("Введіть розмір листа: ");  
 *int* length1 = sc.nextInt();*//вводимо довжину списку* System.***out***.print("Введіть шукане значення: ");  
 *int* search1 = sc.nextInt();  
  
 *for* (*int* i = 0; i < length1; i++) {  
 sList.addFirst((*int*) (Math.*random*() \* 10 - 5));*//заповняємо лист числами* }  
 System.***out***.println(sList);  
  
 System.***out***.println("Завдання згідно варіанту");  
 *taskVar1*(sList);*//викликаємо метод згідно з варіантом* System.***out***.println("Спільне завдання");  
 *task1*(sList, search1);*//викликаємо метод завдання* System.***out***.println(sList);  
  
  
 System.***out***.println("\nЗавдання 2 ");  
 DoublyLinkedList<Integer> dList = *new* DoublyLinkedList<>();  
 System.***out***.print("Введіть розмір листа: ");  
 *int* length2 = sc.nextInt();*//вводимо довжину списку* System.***out***.print("Введіть шукане значення: ");  
 *int* search2 = sc.nextInt();  
  
 *for* (*int* i = 0; i < length2; i++) {  
 dList.addLast((*int*) (Math.*random*() \* 10 - 5));  
 }  
 System.***out***.println(dList);  
  
 System.***out***.println("Завдання згідно варіанту");  
 *taskVar2*(dList);*//викликаємо метод згідно з варіантом  
  
 task2*(dList, search2);*//викликаємо метод завдання* System.***out***.println(dList);  
  
  
 System.***out***.println("\nЗавдання 3 ");  
 System.***out***.print("Введіть розмір стека: ");  
 *int* length3 = sc.nextInt();*//вводимо довжину стеку* Stack<Integer> myStack = *new* Stack<>();*//створюємо стек  
  
 for* (*int* i = 0; i < length3; i++) {  
 myStack.push((*int*) (Math.*random*() \* 10 - 5));  
 }  
 System.***out***.println(myStack);*//виводимо стек* System.***out***.println("Завдання згідно варіанту");  
 *taskVar*(myStack);*//викликаємо метод згідно з варіантом* }  
  
  
 *public static void* task1(SingleLinkedList<Integer> list, Integer key) {*//метод для першого спільного завдання  
  
 for* (*int* i = 1; i < list.size() - 1; i++) {  
  
 *if* (list.get(i).equals(key)) {  
 *if* (list.get(i + 1).equals(key)) {  
  
  
 *int* j = i;  
  
 *while* (list.get(j).equals(key)) {  
 *if* (j == list.size() - 1) *break*;  
  
 list.remove(j);  
  
 }  
 i--;  
 } *else if* (!list.get(i - 1).equals(key)) {  
 *int* temp = list.get(i - 1);  
 list.set(i - 1, list.get(i + 1));  
 list.set(i + 1, temp);  
 list.remove(i);  
 i--;  
  
 } *else* {  
 list.remove(i);  
 i--;  
 }  
 }  
  
 }  
  
 *if* (list.get(0).equals(key)) {  
 list.removeFirst();  
 }  
 *if* (list.get(list.size() - 1).equals(key)) {  
 list.remove(list.size() - 1);  
 }  
  
  
 }  
  
 *public static void* task2(DoublyLinkedList<Integer> list, Integer key) {*//метод для другого спільного завдання  
  
 int* size = list.size();  
 *for* (*int* i = 0; i < size; i++) {  
  
 *if* (list.get(i).equals(key)) {  
 list.remove(i);  
 i--;  
 size--;  
 }  
  
 }  
  
 }  
  
 *public static void* taskVar1(SingleLinkedList<Integer> list) {*//метод для першого завдання згідно з варіантом  
 int* plus = 0;  
 *int* minus = 0;  
  
 *for* (*int* i : list) {  
 *if* (i > 0) {  
 plus += i;  
 } *else* {  
 minus -= i;  
 }  
 }  
  
 System.***out***.println("Сума додатних чисел: " + plus);  
 System.***out***.println("Модуль суми від'ємних чисел: " + minus);  
  
 *if* (plus > minus) {  
 System.***out***.print("\u001B[32m");  
 System.***out***.println("Сума додатних більша за модуль суми від'ємних чисел");  
 } *else if* (plus == minus) {  
 System.***out***.print("\u001B[33m");  
 System.***out***.println("Сума додатних дорівнює модулю від'ємних чисел");  
 } *else* {  
 System.***out***.print("\u001B[31m");  
 System.***out***.println("Модуль суми від'ємних більший за сума додатних чисел");  
 }  
 System.***out***.print("\u001B[0m");  
  
 }  
  
 *public static void* taskVar2(DoublyLinkedList<Integer> list) {*//метод для другого завдання згідно з варіантом  
 int* plus = 0;  
 *int* minus = 0;  
  
 *for* (*int* i : list) {  
 *if* (i > 0) {  
 plus += i;  
 } *else* {  
 minus -= i;  
 }  
 }  
  
 System.***out***.println("Сума додатних чисел: " + plus);  
 System.***out***.println("Модуль суми від'ємних чисел: " + minus);  
 *if* (plus > minus) {  
 System.***out***.print("\u001B[32m");  
 System.***out***.println("Сума додатних більша за модуль суми від'ємних чисел");  
 } *else if* (plus == minus) {  
 System.***out***.print("\u001B[33m");  
 System.***out***.println("Сума додатних дорівнює модулю від'ємних чисел");  
 } *else* {  
 System.***out***.print("\u001B[31m");  
 System.***out***.println("Модуль суми від'ємних більший за сума додатних чисел");  
 }  
 System.***out***.print("\u001B[0m");  
 }  
  
  
 *public static void* taskVar(Stack<Integer> stack) {*//метод для третього завдання згідно з варіантом  
  
 int* plus = 0;  
 *int* minus = 0;  
  
 *for* (*int* i : stack) {  
 *if* (i > 0) {  
 plus += i;  
 } *else* {  
 minus -= i;  
 }  
 }  
  
 System.***out***.println("Сума додатних чисел: " + plus);  
 System.***out***.println("Модуль суми від'ємних чисел: " + minus);  
 *if* (plus > minus) {  
 System.***out***.print("\u001B[32m");  
 System.***out***.println("Сума додатних більша за модуль суми від'ємних чисел");  
 } *else if* (plus == minus) {  
 System.***out***.print("\u001B[33m");  
 System.***out***.println("Сума додатних дорівнює модулю від'ємних чисел");  
 } *else* {  
 System.***out***.print("\u001B[31m");  
 System.***out***.println("Модуль суми від'ємних більший за сума додатних чисел");  
 }  
 System.***out***.print("\u001B[0m");  
  
 }  
}

**Результат виконання програми:**

****

**Висновок:**

На цій лабораторній роботі було ознайомлено з основами роботи з двозв’язним списком, однозв’язним списком, стеком та чергою. Написані відповідні динамічні структури даних. Освоєно основні методи цих структур. Виконано спільні завдання та завдання згідно варіанту. Досліджено результат на правильність