

## Параметризація в Java. Обробка параметризованих контейнерів

### Мета

- Вивчення принципів параметризації в *Java* .
- Розробка параметризованих класів та методів.
- Розширення функціональності параметризованих класів.

### Вимоги

1. Створити власний клас-контейнер, що параметризується ( *Generic Type* ), ([docs.oracle.com/javase/tutorial/java/generics/types.html](https://docs.oracle.com/javase/tutorial/java/generics/types.html)) на основі зв'язних списків для реалізації колекції domain-об'єктів з лабораторної роботи №10 ( Прикладні задачі. Список №2. 20 варіантів )
2. Для розроблених класів-контейнерів забезпечити можливість використання їх об'єктів у циклі `foreach` якості джерела даних.
3. Забезпечити можливість збереження та відновлення колекції об'єктів:
  - 1) за допомогою стандартної серіалізації;
  - 2) не використовуючи протокол серіалізації.
4. Продемонструвати розроблену функціональність: створення контейнера, додавання елементів, видалення елементів, очищення контейнера, перетворення у масив, перетворення у рядок, перевірку на наявність елементів.
5. Забороняється використання контейнерів (колекцій) з *Java Collections Framework*- [docs.oracle.com/javase/8/docs/technotes/guides/collections/](https://docs.oracle.com/javase/8/docs/technotes/guides/collections/)
6. Розробити параметризовані методи ( *Generic Methods*- [docs.oracle.com/javase/tutorial/java/generics/methods.html](https://docs.oracle.com/javase/tutorial/java/generics/methods.html)) для обробки колекцій об'єктів згідно ( Прикладні задачі. Список №2. 20 варіантів ).
7. Продемонструвати розроблену функціональність (створення, управління та обробку власних контейнерів) в діалоговому та автоматичному режимах.
  - a. Автоматичний режим виконання програми задається параметром командного рядка `-auto` . Наприклад, `java ClassName -auto` .
  - b. В автоматичному режимі діалог з користувачем відсутній, необхідні данні генеруються, або зчитуються з файлу

### 1.1 Прикладна задача

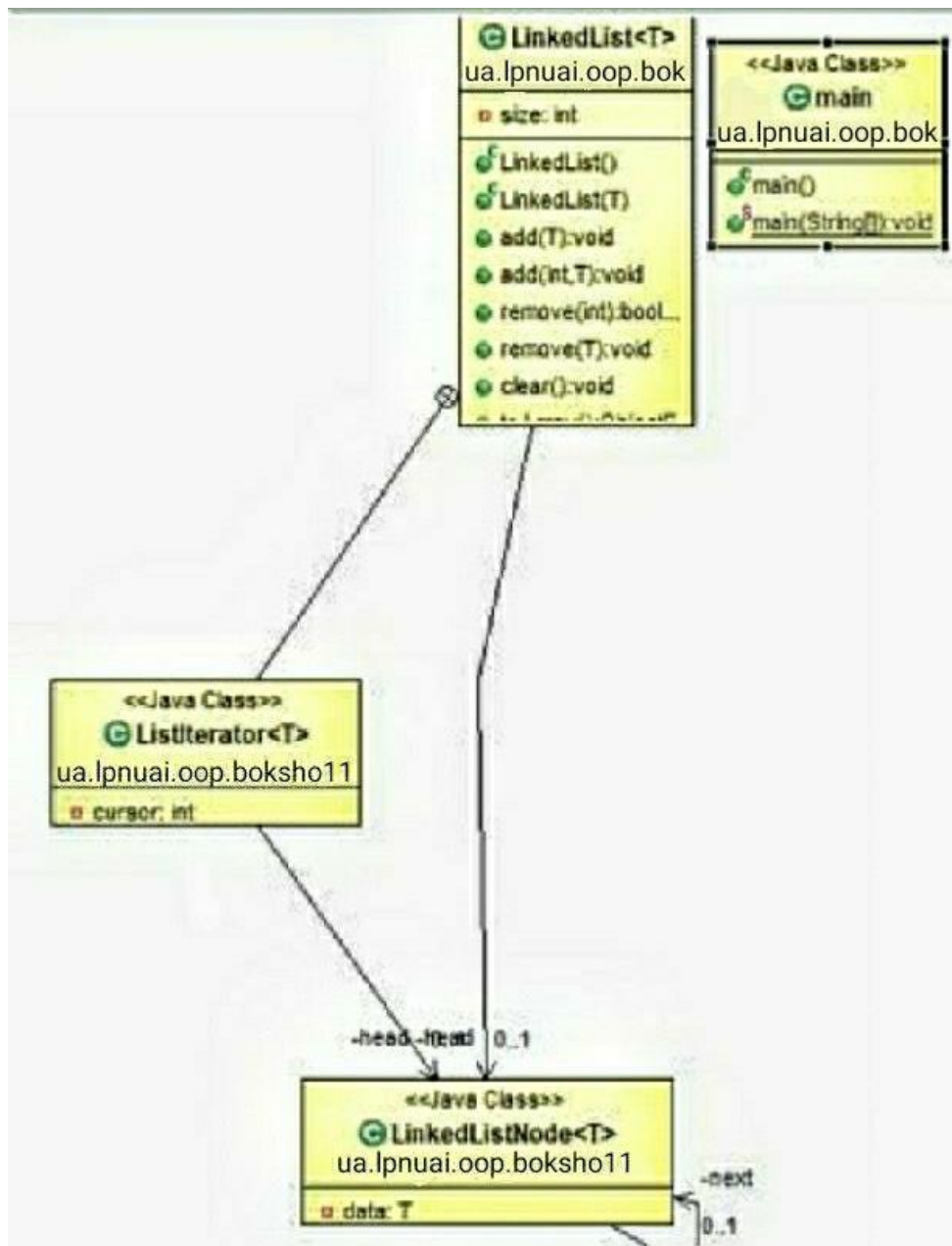
Бюро знайомств. Запис про клієнта: стать; реєстраційний номер; дата реєстрації; відомості про себе (довільний набір властивостей: ім'я, зріст, колір очей, дата народження, хобі тощо); вимоги до партнера (довільний набір властивостей).

## 2. Опис програми

Програма реалізована у вигляді інтерактивного консольного вікна з діалоговим режимом роботи з користувачем. Основне призначення: демонстрація управління масивом domain-об'єктів. Реалізовано додавання та генерування нових об'єктів, видалення, показ інформації.

### 2.1 Ієрархія та структура класів





## 2.2 Важливі фрагменти програми

### (Параметризація в Java)

```

/**
 * Реалізує клієнта бюро знайомств.
 */
public class Client {

    /**
     * Стать
     */
    private String gender;

    /**
     * Реєстраційний номер
     */
    private int regNum;

    /**
     * Дата реєстрації
     */
    private String regDate;
  
```

```
/**
 * Ім'я
 */
private String name;
/**
 * Зріст
 */
private int height;
/**
 * Колір очей
 */
private String eyes;
/**
 * День народження
 */
private String birthday;
/**
 * Хобі
 */
private String[] hobby;
/**
 * Вимоги до партнера
 */
private String[] requirements;

/**
 * Конструктор за замовчуванням
 */
Client() {
    gender = null;
    regNum = 0;
    regDate = null;
    name = null;
    height = 0;
    eyes = null;
    birthday = null;
    hobby = null;
    requirements = null;
}

/**
 * @return the birthday
 */
public String getBirthday() {
    return birthday;
}

/**
 * @return the eyes
 */
public String getEyes() {
    return eyes;
}

/**
 * @return the gender
 */
public String getGender() {
```

```

        return gender;
    }

    /**
     * @return the height
     */
    public int getHeight() {
        return height;
    }

    /**
     * @return the hobbies
     */
    public String[] getHobbies() {
        return hobby;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @return the regDate
     */
    public String getRegDate() {
        return regDate;
    }

    /**
     * @return the regNum
     */
    public int getRegNum() {
        return regNum;
    }

    /**
     * @return the requirements
     */
    public String[] getRequirements() {
        return requirements;
    }

    /**
     * @param birthday
     *         the birthday to set
     */
    public void setBirthday(String birthday) {
        if (birthday == null || birthday.equals("")) {
            throw new IllegalArgumentException(birthday);
        }
        this.birthday = birthday;
    }

    /**
     * @param eyes

```

```

        *           the eyes to set
    */
    public void setEyes(String eyes) {
        if (eyes == null || eyes.equals("")) {
            throw new IllegalArgumentException(eyes);
        }
        this.eyes = eyes;
    }

    /**
     * @param gender
     *           the gender to set
     */
    public void setGender(String gender) {
        if (gender == null || gender.equals("")) {
            throw new IllegalArgumentException(eyes);
        }
        this.gender = gender;
    }

    /**
     * @param height
     *           the height to set
     */
    public void setHeight(int height) {
        if (height <= 0) {
            throw new IllegalArgumentException("" + height);
        }
        this.height = height;
    }

    /**
     * @param hobby
     *           the hobbies to set
     */
    public void setHobbies(String[] hobby) {
        if (hobby.length == 0) {
            throw new IllegalArgumentException(hobby.toString());
        }
        this.hobby = hobby;
    }

    /**
     * @param name
     *           the name to set
     */
    public void setName(String name) {
        if (name == null || name.equals("")) {
            throw new IllegalArgumentException(name);
        }
        this.name = name;
    }

    /**
     * @param regDate
     *           the regDate to set
     */
    public void setRegDate(String regDate) {

```

```

        if (regDate == null || regDate.equals("")) {
            throw new IllegalArgumentException(regDate);
        }
        this.regDate = regDate;
    }

    /**
     * @param regNum
     *         the regNum to set
     */
    public void setRegNum(int regNum) {
        if (regNum <= 0) {
            throw new IllegalArgumentException("" + regNum);
        }
        this.regNum = regNum;
    }

    /**
     * @param requirements
     *         the requirements to set
     */
    public void setRequirements(String[] requirements) {
        if (requirements.length == 0) {
            throw new IllegalArgumentException(requirements.toString());
        }
        this.requirements = requirements;
    }

    /**
     * (non-Javadoc)
     *
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {
        return super.toString();
    }
}

```

## (Обробка параметризованих контейнерів)

```

import java.io.Serializable;
import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 * Список
 * @param <Type>
 *         тип зберігаємих елементів
 */
public class LinkedList<Type> implements Iterable<Type>, Serializable {
    /**
     * Ітератор списку
     *
     * @author student Lytvyn I.I. KIT-26A
     */
    private class LinkedListIterator implements Iterator<Type> {

```

```

        private Node<Type> nextNode;

        public LinkedListIterator() {
            nextNode = head;
        }

        @Override
        public boolean hasNext() {
            return nextNode != null;
        }

        @Override
        public Type next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            final Type res = nextNode.data;
            nextNode = nextNode.next;
            return res;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

    /**
     * Уособлює ланку списку
     *
     * @param <AnyType>
     *         тип елементу
     */
    private static class Node<AnyType> {
        private final AnyType data;
        private Node<AnyType> next;

        public Node(AnyType data, Node<AnyType> next) {
            this.data = data;
            this.next = next;
        }
    }

    /**
     * Унікальний ідентифікатор версії класу
     */
    private static final long serialVersionUID = 3842190203164935628L;

    /**
     * Початок списку
     */
    transient private Node<Type> head;

    /**
     * Розмір списку
     */
    transient private int size;

```



```

/**
 * Constructs an empty list
 */
public LinkedList() {
    head = null;
    size = 0;
}

/**
 * Створює список з іншого списку
 *
 * @param list
 *         список
 */
@SuppressWarnings("unchecked")
public LinkedList(LinkedList<Type> list) {
    head = null;
    size = 0;
    for (final Object x : list) {
        addLast((Type) x);
    }
}

/**
 * Додавання елементу на початок списку
 *
 * @param item
 *         елемент
 */
public void addFirst(Type item) {
    head = new Node<>(item, head);
    size++;
}

/**
 * Додавання елементу в кінець списку
 *
 * @param item
 *         елемент
 */
public void addLast(Type item) {
    if (head == null) {
        size--;
        addFirst(item);
    } else {
        Node<Type> tmp = head;
        while (tmp.next != null) {
            tmp = tmp.next;
        }

        tmp.next = new Node<>(item, null);
    }
    size++;
}

/**
 * Очищення списку
 */

```

```

public void clear() {
    head = null;
    size = 0;
    System.gc();
}

/**
 * Перевіряє чи присутній елемент у списку
 *
 * @param x
 *        елемент для перевірки
 * @return чи присутній елемент у списку
 */
public boolean contains(Type x) {
    for (final Type tmp : this) {
        if (tmp.equals(x)) {
            return true;
        }
    }

    return false;
}

/**
 * Повертає копію списку
 *
 * @return копія списку
 */
public LinkedList<Type> copy() {
    final LinkedList<Type> twin = new LinkedList<>();
    Node<Type> tmp = head;
    if (head == null) {
        return null;
    }
    while (tmp != null) {
        twin.addFirst(tmp.data);
        tmp = tmp.next;
    }

    return twin.reverse();
}

/**
 * Повертає елемент, що знаходиться на заданій позиції
 *
 * @param pos
 *        позиція елементу
 * @return елемент
 */
public Type get(int pos) {
    if (head == null) {
        throw new IndexOutOfBoundsException();
    }

    Node<Type> tmp = head;
    for (int k = 0; k < pos; k++) {
        tmp = tmp.next;
    }

```

```

    }

    if (tmp == null) {
        throw new IndexOutOfBoundsException();
    }

    return tmp.data;
}

/**
 * Повертає перший елемент у списку
 *
 * @return перший елемент списку
 */
public Type getFirst() {
    if (head == null) {
        throw new NoSuchElementException();
    }

    return head.data;
}

/**
 * Повертає останній елемент у списку
 *
 * @return останній елемент списку
 */
public Type getLast() {
    if (head == null) {
        throw new NoSuchElementException();
    }

    Node<Type> tmp = head;
    while (tmp.next != null) {
        tmp = tmp.next;
    }

    return tmp.data;
}

/**
 * Вставляє елемент після заданого
 *
 * @param key
 *            елемент після якого необхідно вставити
 * @param toInsert
 *            елемент для вставляння
 */
public void insertAfter(Type key, Type toInsert) {
    Node<Type> tmp = head;

    while (tmp != null && !tmp.data.equals(key)) {
        tmp = tmp.next;
    }

    if (tmp != null) {
        tmp.next = new Node<>(toInsert, tmp.next);
    }
}

```

```

        size++;
    }

    /**
     * Вставляє елемент перед заданим
     *
     * @param key
     *           елемент перед яким необхідно вставити
     * @param toInsert
     *           елемент для вставляння
     */
    public void insertBefore(Type key, Type toInsert) {
        if (head == null) {
            return;
        }

        if (head.data.equals(key)) {
            addFirst(toInsert);
            return;
        }

        Node<Type> prev = null;
        Node<Type> cur = head;

        while (cur != null && !cur.data.equals(key)) {
            prev = cur;
            cur = cur.next;
        }
        if (cur != null) {
            prev.next = new Node<>(toInsert, cur);
        }
        size++;
    }

    /**
     * @return чи порожній список
     */
    public boolean isEmpty() {
        return head == null && size == 0;
    }

    /**
     * Ітератор
     */
    @Override
    public Iterator<Type> iterator() {
        return new LinkedListIterator();
    }

    /**
     * Видаляє перший знайдений елемент
     *
     * @param key
     *           ключ для пошуку
     */
    public void remove(Type key) {
        if (head == null) {
            throw new RuntimeException("cannot delete");
        }
    }

```

```

    }

    if (head.data.equals(key)) {
        head = head.next;
        return;
    }

    Node<Type> cur = head;
    Node<Type> prev = null;

    while (cur != null && !cur.data.equals(key)) {
        prev = cur;
        cur = cur.next;
    }

    if (cur == null) {
        throw new RuntimeException("cannot delete");
    }

    prev.next = cur.next;
    size--;
}

/**
 * Видалення першого елементу
 *
 * @return видалений елемент
 */
public Type removeFirst() {
    final Type tmp = getFirst();
    head = head.next;
    return tmp;
}

/**
 * Змінює порядок розташування елементів у списку на обернений
 *
 * @return обернений список
 */
public LinkedList<Type> reverse() {
    final LinkedList<Type> list = new LinkedList<>();
    Node<Type> tmp = head;
    while (tmp != null) {
        list.addFirst(tmp.data);
        tmp = tmp.next;
    }
    return list;
}

/**
 * @return розмір списку
 */
public int size() {
    return this.size;
}

/**
 * Конвертує список у масив об'єктів

```

```

*
* @return масив об'єктів
*/
public Object[] toArray() {
    final Object[] copy = new Object[this.size];
    int i = 0;
    for (final Object object : this) {
        copy[i++] = object;
    }
    return copy;
}

/**
 * Returns a string representation
 *
 */
@Override
public String toString() {
    final StringBuffer result = new StringBuffer();
    for (final Object x : this) {
        result.append(x + " ");
    }

    return result.toString();
}

/**
 * Відновлює екземпляр <tt>LinkedList</tt> з потоку (тобто десеріалізує
 * його).
 *
 * @param inStream
 *             потік, з якого відновлюється екземпляр <tt>LinkedList</tt>
 * @throws java.io.IOException
 *             виключна ситуація при введенні/виведенні
 * @throws ClassNotFoundException
 *             виключна ситуація при відсутності необхідного класу
 */
@SuppressWarnings("unchecked")
private void readObject(java.io.ObjectInputStream inStream)
    throws java.io.IOException, ClassNotFoundException {
    /* Зчитування розміру та інших прихованих речей */
    inStream.defaultReadObject();
    /* Зчитування довжини масиву */
    final int size = inStream.readInt();

    /* Initialize header */
    head = new Node<>(null, null);

    /* Зчитування всіх елементів у належному порядку */
    for (int i = 0; i < size; i++) {
        addLast((Type) inStream.readObject());
    }
}

/**
 * Зберігає екземпляр <tt>LinkedList</tt> в потоці (тобто серіалізує
 * його).
 *

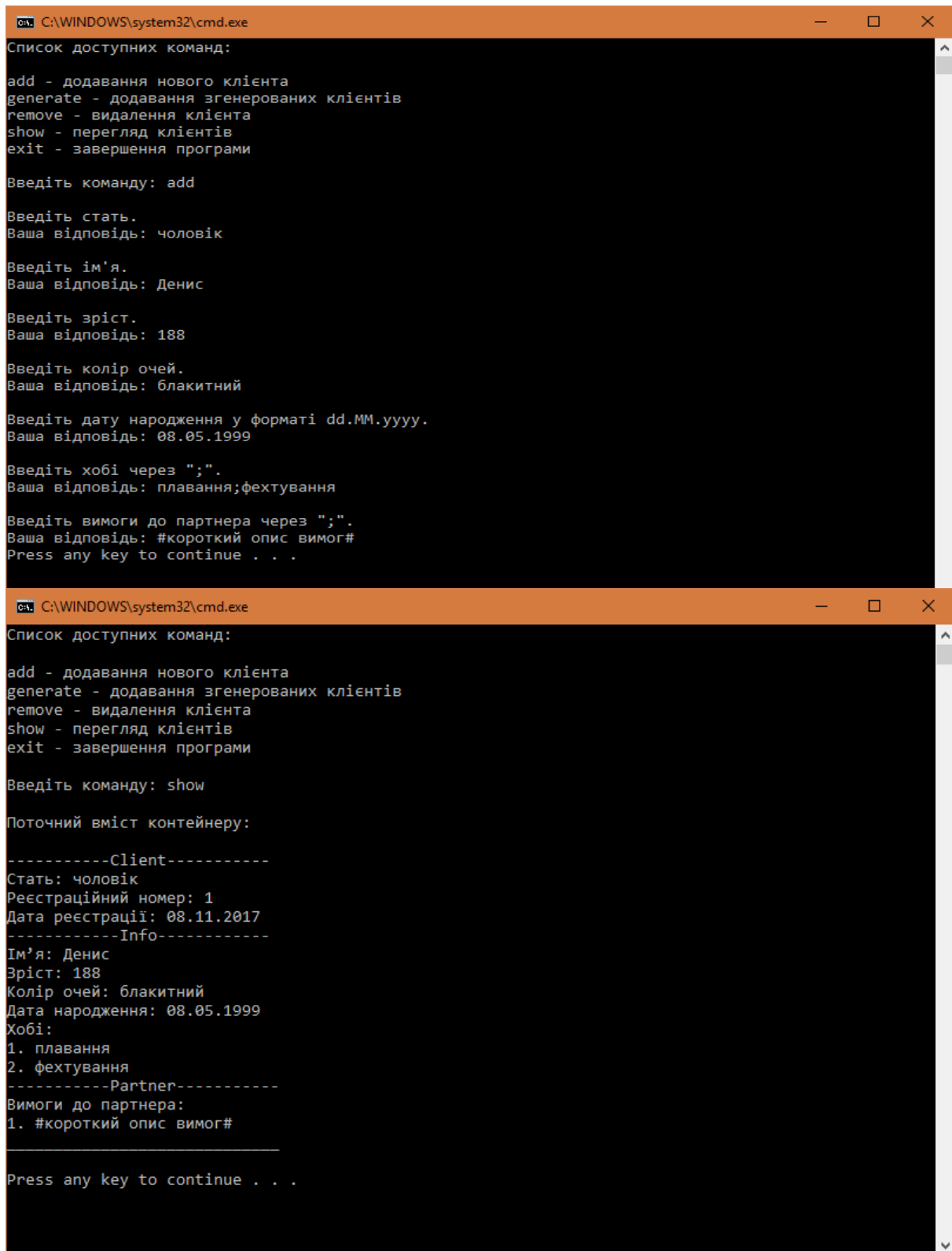
```

```

* @param outputStream
*      потік, в який записується екземпляр <tt>LinkedList</tt>
* @throws java.io.IOException
*      виключна ситуація при введенні/виведенні
*/
private void writeObject(java.io.ObjectOutputStream outputStream)
    throws java.io.IOException {
    /* Запис кількості елементів та інших прихованих речей */
    outputStream.defaultWriteObject();
    /* Запис довжини масиву */
    outputStream.writeInt(size);
    /* Запис всіх елементів у належному порядку. */
    Node<Type> tmp = head;
    while (tmp != null) {
        outputStream.writeObject(tmp.data);
        tmp = tmp.next;
    }
}
}

```

# РЕЗУЛЬТАТ РОБОТИ



```
C:\WINDOWS\system32\cmd.exe
Список доступних команд:
add - додавання нового клієнта
generate - додавання згенерованих клієнтів
remove - видалення клієнта
show - перегляд клієнтів
exit - завершення програми

Введіть команду: add

Введіть статть.
Ваша відповідь: чоловік

Введіть ім'я.
Ваша відповідь: Денис

Введіть зріст.
Ваша відповідь: 188

Введіть колір очей.
Ваша відповідь: блакитний

Введіть дату народження у форматі dd.MM.yyyy.
Ваша відповідь: 08.05.1999

Введіть хобі через ";".
Ваша відповідь: плавання;фехтування

Введіть вимоги до партнера через ";".
Ваша відповідь: #короткий опис вимог#
Press any key to continue . . .

C:\WINDOWS\system32\cmd.exe
Список доступних команд:
add - додавання нового клієнта
generate - додавання згенерованих клієнтів
remove - видалення клієнта
show - перегляд клієнтів
exit - завершення програми

Введіть команду: show

Поточний вміст контейнеру:

-----Client-----
Стать: чоловік
Реєстраційний номер: 1
Дата реєстрації: 08.11.2017
-----Info-----
Ім'я: Денис
Зріст: 188
Колір очей: блакитний
Дата народження: 08.05.1999
Хобі:
1. плавання
2. фехтування
-----Partner-----
Вимоги до партнера:
1. #короткий опис вимог#

Press any key to continue . . .
```

## ВИСНОВКИ

Створено і налагоджено програму, що повністю виконую поставлене індивідуальне завдання та відповідає вимогам. Було отримано і вдосконалено навички у використанні об'єктно-орієнтованого підходу для розробки об'єкта предметної (прикладної) галузі.