УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Кафедра інформаційних технологій та програмної інженерії

КУРСОВА РОБОТА

з Об’єктно-орієнтованого програмування

на тему: «Клас для обслуговування структур типу “Черга”»

Студента II курсу____групи

галузі знань 12 «Інформаційні технології»

спеціальності 121 «Інженерія програмного забезпечення»

Пашника А.П.

(прізвище та ініціали)

Керівник _____к.т.н., Пашкевич О.П.

Національна шкала _____

Кількість балів _____ Оцінка: ECTS _____

м. Івано-Франківськ

2021

Бланк завдання на курсовий проект (роботу)

_____
назва вищого навчального закладу

Кафедра _____

Дисципліна _____

Спеціальність _____

Курс_____ Група _____ Семестр _____

ЗАВДАННЯ

на курсовий проект (роботу) студента

_____
Прізвище, ім'я, по-батькові

1. Тема проекту (роботи)

_____

_____

_____

2. Строк здачі студентом закінченого проекту (роботи)

_____

_____

_____

3. Вихідні дані до проекту (роботи) _____

_____

_____

4. Зміст роботи (перелік питань по розділах, які потрібно розробити)

_____

_____

_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

____

_____

_____

6. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсового проекту (роботи)	Строк виконання	Примітки

Студент _____ _____

Підпис Прізвище, ініціали

Керівник _____ _____
Підпис Прізвище, ініціали

«____» _____ 202___ р.

ЗМІСТ

ВСТУП

Черга- це є структура даних, котра, так як і стек, має свої обмеження в додаванні і відніманні елементів. Щоб зрозуміти цю структуру даних, необхідно уявити собі чергу в магазин: людина, яка перша прийшла буде обслуговуватися першою(оскільки вона ж першою і прийшла), люди які прийшли наступними-стають в кінець черги. Таким чином, людина яка потрапила до каси першою, буде і обслуговуватися відповідно першою, той ж який і добрався до каси останній, буде останнім і обслуговуваний. Черги досить часто використовуються в програмуванні різних мереж, операційних системах, черги пісень у різних Інтернет-сервісах потокового аудіо (таких як Spotify, Apple Music, YouTube Music, Deezer та інші), сервіси доставок, та в багатьох інших речах. У реальному житті, концепція черги здається простою, але в програмуванні з чергою не все так солодко. Повернемося до прикладу черги у супермаркеті. Припустімо, що один покупець покидає чергу. А що ж далі? Усі учасники черги повинні зайняти місце свого ж попередника. Тепер уявімо собі, що тільки один покупець може зробити крок уперед одночасно, щоб зайняти місце свого попередника. Тобто другий покупець займає місце першого, потім

третій займає- другого, і тільки тоді четвертий займає місце третього, і так до останнього покупця. А що, якщо б ніхто не міг вийти або зайти в чергу, поки всі не зайняли місце свого попередника. Звідси можна зрозуміти, що черга буде рухатися дуже і дуже повільно.

Моє завдання полягає в тому щоб розробити демонстраційно-тестуючу програму, яка міститиме у собі класовий тип для обслуговування структур типу “Черга”. А також створити відповідні конструктори для того, щоб було можливо створити об’єкти динамічного типу у програмі. Ще необхідно реалізувати компонентні методи для обслуговування структур типу “Черга”, які вказані в “Таблиці 1”.

Таблиця 1 – Методи, які необхідно реалізувати в програмі.

Назва методу	Реалізація
equals()	Порівняння черг
get()	Доступ по індексу
size()	Кількість елементів
empty()	Повертає значення true, якщо черга пуста
front()	Посилання на перший елемент
back()	Посилання на останній елемент
swap()	Обмін значеннями з іншою чергою
push()	Додати новий елемент у кінець черги
pop()	Вилучити перший елемент
find()	Перевірити, чи є наданий елемент у черзі
accumulate()	Накопичення суми або добутку
for_each()	Обробка елементів по наданій процедурі
max(),min()	Пошук максимального та мінімального елементів

АНАЛІЗ СТАНУ ПРОГРАМУВАННЯ

У світі існує безліч мов програмування C++, C#, JavaScript, та безліч інших які підтримують ООП. Проте ця курсова робота буде розроблена однією із найпопулярніших мов програмування, а саме - мовою Java. Дана мова програмування була створена у 1995 році, компанією Sun Microsystems, проте згодом у 2009 році перейшла до власності компанії Oracle.

Дана мова програмування мала не завжди таку звичну для всіх назву. Спочатку, при створенні їй дали ім’я Oak, що з англійської мови перекладається як “Дуб”. Java отримала таку назву напочатку, тому що під вікном офісу Джеймса Гослінга ріс дуб, який подобався дуже йому, тому в честь дуба він вирішив назвати нову мову програмування. Проте коли компанія Sun Microsystems, побачила що ця мова дуже добре себе відрекомендувала, то ж вони вирішили її перейменувати, оскільки таке ім’я вже носила трохи інша мова програмування, і

необхідно було ім’я, яке було б милозвучніше і комерційніше, тож нова мова програмування у 1995 році отримала нове своє імення, а саме - Java.

Існують 3 основні версії, чому мова програмування з іконкою кави, отримала саме таку назву:

1. Усі програмісти часто п’ють дуже багато кави. Поки люди створювали Java, програмісти випили тералітри кави, тому було вирішено назвати мову в честь відомого сорту кави - Java.
2. Названо в честь марки кави Java, яку назвали в честь острова Ява.
3. Оскільки нову мову програмування спочатку планували розробити спеціально для побутових приладів, то інколи це пов’язують з посиланням на кавову машину, як прилад побутового приладу.

Ця мова програмування є абсолютно об’єктно-орієнтованою. Об’єктно-орієнтоване програмування - це одна із основних парадигм програмування, котра бачить програму, як множину об’єктів, що взаємодіють між собою. ООП містить у собі чотири основних механізми-принципи :

- Абстракція – це суттєві характеристики якогось об’єкту, які відділяють його від решти видів об’єктів, і таким же чином визначають особливості цього об’єкту з точки зору подальшого розгляду та аналізу. Вона зосереджується на суттєвих з погляду спостерігача характеристиках об’єкта
- Інкапсуляція – це властивість системи, яка дозволяє об’єднати дані і методи, які працюють з ними в класі, і приховати деталі реалізації від користувачів. Напряму доступ до стану об’єкту буде заборонено, і з зовні з ним можна буде взаємодіяти тільки за допомогою інтерфейсу. Програмісти зможуть як завгодно редагувати всі закриті елементи, не переживаючи, що хтось їх буде використовувати у своїх програмах.

- Наслідування – це одна із найважливіших абстракцій, яка дає здібність об’єкту або класу успадковувати ті чи інші методи або властивості від іншого класу. Новий клас(дочірній), який створений на основі існуючого класу, може отримати властивості та поведінку від батьківського класу(існуючого), а також доповнити їх своїми ж власними. Множинне успадкування можна втілити у життя за допомогою інтерфейсів.
- Поліморфізм – реалізація завдань, які мають ідентичну ідею, різними способами. Його великим привілеєм є те, що він допомагає зменшувати складність програми, дозволяючи при цьому використання того ж інтерфейсу для завдання єдиного класу дій. Вибір же конкретної дії, в залежності від ситуації, покладається на компілятор.

На сьогоднішній день, дуже багато мов підтримують ООП, або ж загалом самі ж є об’єктно-орієнтованими, наприклад C#, PHP, C++, Python, раніше згадана Java та багато інших. Народження ООП відкрила програмістам можливість працювати з програмами більшого обсягу, оскільки полегшувала промислове програмування, і також створення різних додатків.

Мови, що підтримують ООП, завжди використовують успадкування для повторного використання коду та розширення класу або поведінки об’єкту. У ООП є дві основні концепції класів:

- Об’єкт – це змінна типу клас.
- Клас – описує дані і методи, які використовуватимуться об’єктом цього ж класу.

ПОСТАНОВКА І РОЗРОБКА ПРОГРАМИ ВИКОНАННЯ ЗАВДАННЯ

ЗАГАЛЬНО ПРО ЧЕРГИ

В Об’єктно-орієнтованому програмування також існує таке поняття, як “Черга”. Черга(англійською Queue)- це дуже цікавий тип зберігання даних в програмуванні. З нею ми можемо проводити обмежену кількість операцій. Ці операції ми зазвичай проводимо з верхнім або першим елементом, і не маємо

доступу до середини черги. В реальному житті, ми не можемо прийти і одразу стати в середину черги, або на початок, хіба що у випадку, коли черга пуста. Люди завжди стають у кінець черги. Так само працює черга і в програмуванні. Інколи бувають такі моменти, що не завжди можна ввійти в Queue, оскільки в програмуванні черга має обмежену кількість місць. В реальному житті можна навести такі приклади, коли черга доходить до тебе і враз - починається обідня перерва на касі або ж каса тимчасово не працює.

В програмуванні є декілька тип черг:

- Priority Queue – це черга з пріоритетом. Ця черга цікава тим, що в ній елементи розміщуються не тільки по порядку, але і за пріоритетом. Цей пріоритет можна задавати самому.
- Queue – це звичайний тип черги, яку ми розглядали вище. Працює за принципом “FIFO”, яке розшифровується, “First In – First Out”. За таким же принципом працює і каса в кінотеатрі, перший в черзі буде першим покупцем квитка. За такою темою, буде розроблена ця курсова робота.
- Stack- тип черги, який протилежний Queue, який працює за принципом “LIFO”. Ця абревіатура розшифровується, як “Last In – First Out”, що перекладається як “Останній ввійшов – перший вийшов”. В реальному житті нагадує дитячу, іграшкову піраміду з бубликами різних розмірів.
- Deque – це форма черги, яка містить у собі властивості Queue, та Stack. Розшифровується дек, як Double ended queue- черга з двома кінцями.

Зазвичай у програмуванні черга, має свої власні методи, якими вона може додавати і забирати елементи з черги. В черзі ці методи називаються, add() – метод, який додає елемент в кінець черги; remove() – це метод, який видаляє верхній/перший елемент з черги; peek() – це метод, котрий повертає верхній/перший елемент, також над чергою можна виконувати такі операції, як перевірка на пустоту черги, Черги можуть мати елементи будь якого типу даних. Про те у черги є і свої недоліки. Один з цих недоліків – це відсутність індексів.

Якщо ми б хотіли видалити, якийсь елемент у середині черги, то зразу напряму ми цього зробити не зможемо. Видалення певного елементу, залежить від видалення попереднього елементу. Щоб видалити якийсь елемент із середини черги, потрібно видалити і усі попередні елементи із поточної черги.

Черга складається з голови(head), та хвоста(body) черги. До голови належить перший елемент у списку, до хвоста- усі решта. Видалення з черги елементів відбувається зі сторони голови, додавання до черги відбувається з протилежної ж сторони – з хвоста, так як зображена на Рис. 2.2.1

Рис. 2.2.1

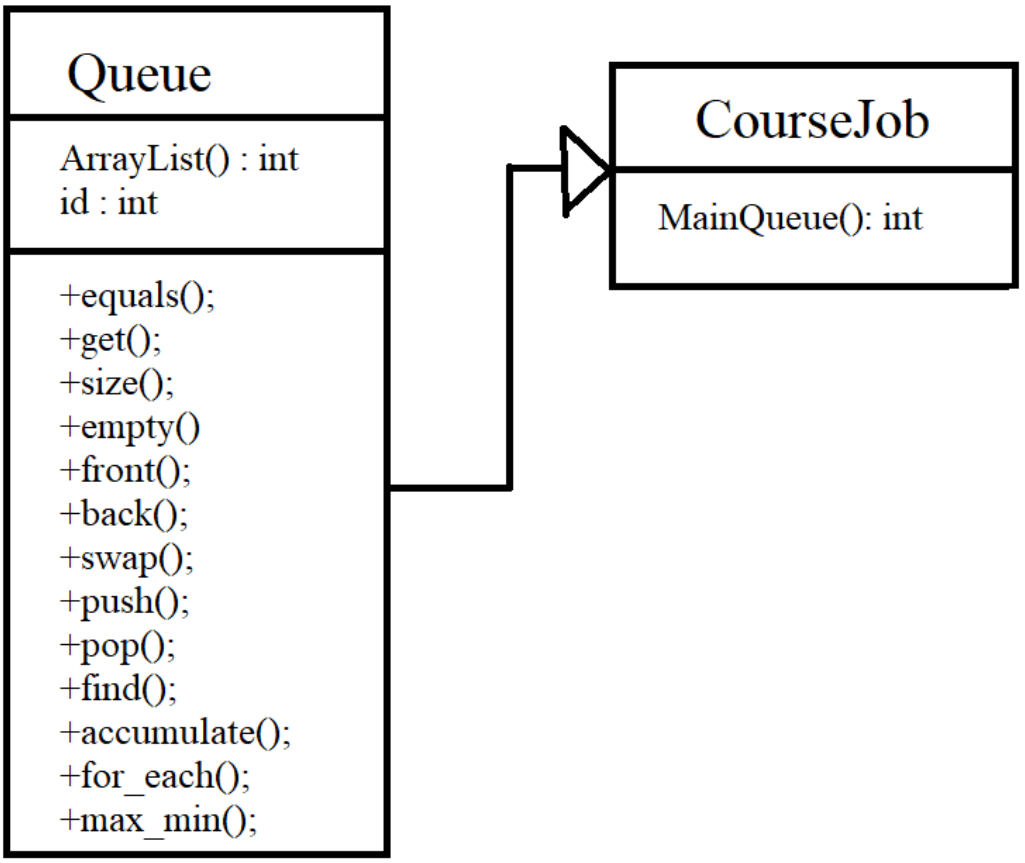
Створення пустої черги полягатиме у тому, що буде потрібно присвоїти голові і хвосту нульові значення, що буде означати, що черга є пустою.

ТЕОРЕТИЧНА РОЗРОБКА ПРОГРАМИ

У даній програмі будуть створені два класи, один з яких буде називатися Queue. Цей клас міститиме у собі всі необхідні методи, щоб вказувати на потрібну

поведінку класу “CourseJob” . На початку необхідно створити клас Queue, у якому ми створимо чергу з цілочисельними значеннями та методами, які необхідно реалізувати в черзі: equals() щоб порівнювати черги на рівність, get() щоб отримувати доступ до елементів по індексу, size()- дізнаватися розмір черги, empty()-щоб взнавати, чи черга часом не пуста, front() - який б давав посилання на перший елемент, back() - котрий би давав посилання на останній елемент swap() – який би обмінював значення з іншою чергою, push() – що додає елементи в кінець черги, pop()- вилучає перший елемент, find() що знаходитиме елементи за їхнім індексом, accumulate() який накопичуватиме суму елементів, for_each() що має обробляти елементи по наданій процедурі, max_min() що шукатиме максимальні та мінімальні елементи. Оскільки ми будемо працювати з індексами елементів, то нам знадобиться відповідна структура даних, в яких елементи будуть впорядковані, порядок яких буду визначатися індексами або вказівниками, під цю задачу підходив би масив, але на жаль він не є безрозмірним. Отож використовуватимемо зв’язний список, який є динамічним і дозволяє звертатися до елементів за індексами. Також нам необхідно створити другий клас CourseJob, який буде викликати методи з попердньо створеного класу. Приклад можна побачити на UML-діаграмі, яка зображена на Рис.2.2

Рис.2.2 UML-діаграми



РОЗРОБКА ПРОГРАМИ

Для початку необхідно створити клас Queue, у цьому класі як і попередньо було сказано, будуть реалізовуватися усі попередньо згадані методи.

```
public int get(int id) {
   ArrayList.get(id);
   System.out.println("Під індексом "+id+" знаходиться елемент "+
ArrayList.get(id));
   return ArrayList.get(id);
}
```

//Ця частина коду створює метод get, який приймає цілочисельні значення індексу, виводить в консоль елемент, який знаходиться під таким індексом, а також повертає його;

```
public void size() {
   int SizeOfArray = 0;
   for (int id = 0; id < ArrayList.size(); id++) {
         SizeOfArray++;
   }
   System.out.println("Черга складається з " + SizeOfArray + " елементів.");
}
```

// Тут створено метод size. Цей метод за допомогою циклу for метод рахує кількість елементів у нашій черзі, та виводить його у консоль програми;

```
public boolean empty() {
   if (ArrayList.isEmpty()) {
      System.out.println("Черга пуста");
```

```
        return true;
    } else {
        return false;
    }
}
```

//В створеному методі empty реалізована умова, яка перевіряє чи є в черзі хоча б якісь елементи. Якщо у черзі немає жодного елементу (вона пуста), то вона виводить у консоль повідомлення, про те що черга є пустою і повертає значення true, якщо ж у черзі присутні якісь елементи, то – false;

```
public void max_min() {
    int Max = -999;
    int Min = 999;
    int MaxID = 0;
    int MinID = 0;
    for (int id = 0; id < ArrayList.size(); id++) {
        if (ArrayList.get(id) < Min) {
            Min = ArrayList.get(id);
            MinID = id+1;
        }
        if (ArrayList.get(id) > Max) {
            Max = ArrayList.get(id);
            MaxID = id+1;
        }
    }
    System.out.println("Мінімальне значення " + Min + " під номером " + MinID);
    System.out.println("Максимальне значення " + Max + " під номером " +
```

```
MaxID);
}
```

//Даний блок коду з методом max_min за допомогою циклу for проганяє усі елементи у черзі по індексу і за допомогою функції умови if шукає мінімальні та максимальні значення, в кінці блоку прописана функція, яка каже програмі вивести в консоль найменше та найбільше значення з їхніми індексами у консольний рядок;

```
  public Integer front() {
    int id = 0;
    do {
        id++;
    } while (ArrayList.get(id) == null);
    return ArrayList.get(id);

}
```

//Даний метод front, за допомогою циклу DoWhile, дає посилання на перший елемент у черзі і також повертає його;

```
public Integer back() {
    return ArrayList.lastIndexOf(ArrayList);
}
```

//Даний метод back повертає останній елемент у черзі;

```
public int push(int element) {
    int i;
```

```
    int maxID = 0;
    for (i = 0; i < ArrayList.size(); i++) {
        maxID = i;
    }
    ArrayList.add(maxID + 1, element);
    return ArrayList.size();
}
```

//Даний метод push приймає елемент який потрібно вставити в кінець черги. За допомогою циклу for, знаходить останній елемент у черзі і ставить заді нього елемент який метод прийняв. Також цей метод повертає розмір масиву;

```
public boolean equals() {
    if (ArrayList == ArrayList2) {
        System.out.println("Черги рівні");
        return true;
    } else {
        System.out.println("Черги не є рівними");
        return false;
    }
}
```

//Метод equals порівнює дану чергу з попередньо заготовленою чергою у випадку, коли обидві черги рівні - на консолі буде виведено повідомлення, що ці черги є рівними. У протилежному ж випадку виведеться повідомлення про те що дані черги не є рівними;

```
public void pop() {
    ArrayList.remove(0);
}
```

//Метод pop видаляє перший елемент з черги;

```
public void for_each() {
    for (int OutArrayList : ArrayList) {
        System.out.println(OutArrayList);
    }
}
```

//Метод for_each проходиться по кожному елементу в черзі і виводить його у консоль;

```
public void accumulate() {
    int sum = 0;
    for (int id = 0; id < ArrayList.size(); id++) {
        sum += ArrayList.get(id);
    }
    System.out.println("Сума всіх елементів дорівнює " + sum);
}
```

//Даний метод проходиться по всій черзі і знаходить суму всієї черги

```
public void find(int FindMe) {
    boolean SayTrue=false;
    for (int id = 0; id < ArrayList.size(); id++) {
        if (FindMe == ArrayList.get(id)) {
            SayTrue = true;
```

```
        } else {
            SayTrue = false;


        }
    }
        if (SayTrue == true) {
            System.out.println("Наданий елементі існує у черзі");
        } else {
            System.out.println("Наданого елементу не існує у черзі");
        }
}
```

У класі CourseJob, необхідно створити метод, який би викликав усі попередньо створені методи…

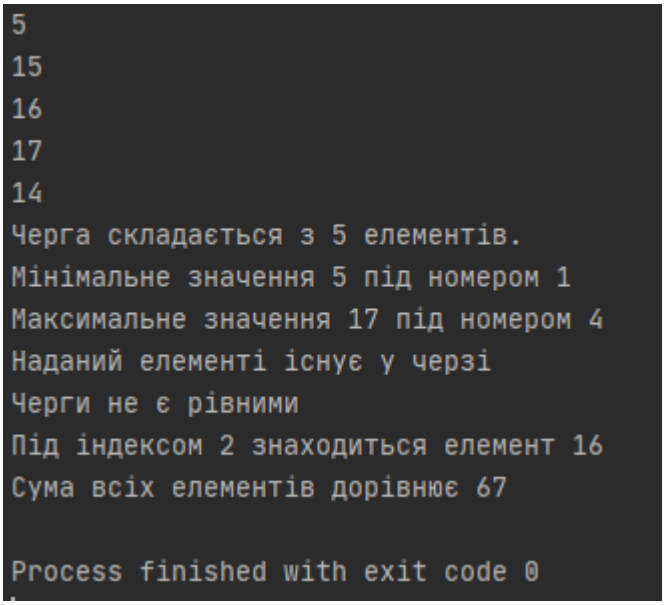
```
public static void main(String[] args) {

    Queue MainQueue = new Queue();
    MainQueue.push(15);
    MainQueue.push(16);
    MainQueue.push(17);
    MainQueue.push(14);
    MainQueue.pop();
    MainQueue.for_each();
    MainQueue.size();
    MainQueue.max_min();
    MainQueue.find(14);
    MainQueue.equals();
    MainQueue.get(2);
    MainQueue.empty();
```

```
    MainQueue.front();
    MainQueue.back();
    MainQueue.accumulate();
}
```

// Даний метод main створює нову чергу на основі попереднього класу Queue. Також цей метод викликає всі попередньо створені методи, для того, щоб вони виконали свої функції які ми прописували раніше.

В результаті ми отримаємо приблизно таку відповідь, яка зображена на Рис.3.1

Рис.3.1



Висновок

Під час виконання даної роботи, я навчився застосовувати ООП для написання програми “Queue”. Було відпрацьовано основні методи та засоби об’єктно-орієнтованого програмування. На цій курсовій роботі було розроблено з нуля структуру даних “Черга”. Яка порівнює черги між собою, надає користувачу доступ по індексу, показує користувачу розмір черги, чи черга пуста чи ні, дає посилання на перший та останній елементи. Також вона дозволяє додати елемент у кінець черги та вилучити перший елемент черги. Перевірити чи є в існує наданий елемент у черзі. Також ця конструкція дозволяє накопичувати суму всіх елементів, виводить всі елементи черги у консоль і шукає максимальний та мінімальний елементи. Під час даної курсової роботи вдалося реалізувати усі компонентні методи, які були включені у завдання

курсової мого варіанту. Також в процесі виконання курсової роботи було закріплено знання з Об’єктно-Орієнтованого Програмування. Для створення динамічної черги, я взяв за основу таку структуру даних, як зв’язаний-список. Цим же ми зробили роботу коду трохи довшою, ніж час роботи структури даних “Черга”. Дякуючи цій курсовій роботі, було також закріплені знання з UML-діаграм, та і самого ООП. Набув практичних навиків роботи. Покращив свої вміння програмування мовою Java, знання з ООП і UML- діаграм. Дана черга, може нагадувати врізану структуру даних Stack, оскільки також має доступ до перегляду перших і останніх елементів. Але не має доступу до видаляння елементів з кінця та початку черги. Дана черга частково схожа на звичайний масив, оскільки останній також дозволяє користувачу звертатися до елементів за допомогою індексів. Оскільки масив має обмежену кількість місць і його не можна розширити, було прийнято рішення використовувати, як фундамент- зв’язаний список. Оскільки дана структура даних була динамічною і в неї можна було в процесі виконання коду додавати нові елементи. Мова програмування Java, є сама по собі об’єктно-орієнтованою. В основу ООП входять такі фундаментальні компоненти: абстракція, інкапсуляція, поліморфізм та успадкування. ООП має декілька своїх принципів. Головний принцип його це те, що все є обєктами. Також в Java у класі задається поведінка об’єкта, яка дозволяє усім об’єктам, що є екземплярами одного класу, виконувати ідентичні дії

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Эккель Б. Философия Java.: 4-е изд. – СПб.: Питер, 2009. – 640 с.

2. Хорстманн К. С., Корнелл Г. Библиотека профессионала. Java 2. Том 1. Основы, 7-е изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2007. – 896 с.

3. Хорстманн, К., С., Корнелл, Г. Библиотека профессионала. Java 2. Том 2. Тонкости программирования, 7-е изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2007. – 1168 с.

4. Шилдт Г. Полный справочнык по Java. Java SE 6 Edition, 7-е изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2007. – 1034 с.

5. Флэнаган Д. Java. Справочник, 4-е изд.: Пер. с англ. – Спб.:

6. Абельсон Х. Структура и интерпретация компьютерных программ / Абельсон Х., Джеральд Дж. С., Сассман Дж. / М.: Добросвет, 2006. – 608 с.

7. Хорстманн К. С., Корнелл Г. Библиотека профессионала. Java 2. Том 1. Основы, 7-е изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2007. – 896 с.

8. Хорстманн, К., С., Корнелл, Г. Библиотека профессионала. Java 2. Том 2. Тонкости программирования, 7-е изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2007. – 1168 с.

9. Зубов В.С. Справочник программиста. Базовые методы решения графовых задач и сортировки. – М.: "Филинъ", 1999. – 256 с.

10. В.В.Белов, Е.М.Воробьев, В.Е.Шаталов. Теория графов. – М.: Высш. шк., 1976. – 392 с.

11. Оре О. Теория графов. – М.: 1980.

12. А.А.Зыков. Основы теории графов.- М.: Наука, 1987. – 384 с.

13. Ю.Н.Тюрин, А.А.Макаров. Анализ данных на компьютере. – М.: ИНФРА-М, Финансы и стат., 1995. – 384 с.

14. Л.И.Турчак. Основы численных методов. – М.: Наука, 1987. – 320 с.

15. В.Н.Нефедов, В.А.Осипова. Курс дискретной математики: Учеб. пос. – М.: Изд-во МАИ, 1992. – 264 с.

16. Bloch J. Effective Java: Programming Language Guide. Publisher: Addison Wesley, 2001. – 180 p.

17. Wikipedia “Об’єктно-орієнтоване програмування”.

18. Хабр “ООП с примерами (часть 2)”.

19. Vertex Academy: “Что такое ArrayList в Java”.

20. Портал освітньо-інформаційних послуг «Студентська консультація»

21. Портал allbest: “Розробка програм мовою С++ з використанням технології об’єктно-орієнтованого програмування”;

22. Портал JavaRush “UML что это?”;

23. Wikipedia “Черга(абстрактний тип даних)”.

24. Портал JavaRush “Queue в Java”

25. Портал StudMe.org “ArrayList”

26. Татьяна Павловская C/C++. Процедурное и объектно-ориентированное программирование. Учебник / Татьяна Павловская. - М.: Питер, 2015. - 496 с.

27. Дж. Кьоу Объектно-ориентированное программирование / Дж. Кьоу, М. Джеанини. - М.: Питер, 2015. - 240 с.

28. Н.Н. Мартынов Алгоритмизация и основы объектно-ориентированного программирования на JavaScript. Информатика и ИКТ. Профильный уровень. 10 класс / Н.Н. Мартынов. - Москва: Высшая школа, 2014. - 272 с.

ДОДАТКИ

Додаток А

Текст програми

```
import java.util.ArrayList;
import java.util.Arrays;

class Queue {
   public ArrayList<Integer> ArrayList = new ArrayList<Integer>(Arrays.asList(1,
5));
   int id = 0;
   public ArrayList<Integer> ArrayList2 = new ArrayList<>();

   public int get(int id) {
      ArrayList.get(id);
      System.out.println("Під індексом " + id + " знаходиться елемент " +
ArrayList.get(id));
      return ArrayList.get(id);

   }

   public void size() {
      int SizeOfArray = 0;
      for (int id = 0; id < ArrayList.size(); id++) {
         SizeOfArray++;
      }
      System.out.println("Черга складається з " + SizeOfArray + " елементів.");
   }
```

```
    public boolean empty() {
        if (ArrayList.isEmpty()) {
            System.out.println("Черга пуста");
            return true;
        } else {
            return false;
        }
    }

    public void max_min() {
        int Max = -999;
        int Min = 999;
        int MaxID = 0;
        int MinID = 0;
        for (int id = 0; id < ArrayList.size(); id++) {
            if (ArrayList.get(id) < Min) {
                Min = ArrayList.get(id);
                MinID = id + 1;
            }
            if (ArrayList.get(id) > Max) {
                Max = ArrayList.get(id);
                MaxID = id + 1;
            }
        }
        System.out.println("Мінімальне значення " + Min + " під номером " +
MinID);
        System.out.println("Максимальне значення " + Max + " під номером " +
```

```
MaxID);
    }

    public void accumulate() {
        int sum = 0;
        for (int id = 0; id < ArrayList.size(); id++) {
            sum += ArrayList.get(id);
        }
        System.out.println("Сума всіх елементів дорівнює " + sum);
    }

    public void find(int FindMe) {
        boolean SayTrue=false;
        for (int id = 0; id < ArrayList.size(); id++) {
            if (FindMe == ArrayList.get(id)) {
                SayTrue = true;
            } else {
                SayTrue = false;


            }
        }
            if (SayTrue == true) {
                System.out.println("Наданий елементі існує у черзі");
            } else {
                System.out.println("Наданого елементу не існує у черзі");
            }
```

```
}

public Integer front() {
    int id = 0;
    do {
        id++;
    } while (ArrayList.get(id) == null);
    return ArrayList.get(id);

}

public Integer back() {
    return ArrayList.lastIndexOf(ArrayList);
}


public int push(int element) {
    int i;
    int maxID = 0;
    for (i = 0; i < ArrayList.size(); i++) {

        maxID = i;
    }
    ArrayList.add(maxID + 1, element);
    return ArrayList.size();
}

public boolean equals() {
```

```
        if (ArrayList == ArrayList2) {
            System.out.println("Черги рівні");
            return true;
        } else {
            System.out.println("Черги не є рівними");
            return false;
        }
    }

    public void pop() {
        ArrayList.remove(0);
    }


    public void for_each() {

        for (int OutArrayList : ArrayList) {
            System.out.println(OutArrayList);
        }
    }


}

class CourseJob {
    public static void main(String[] args) {

        Queue MainQueue = new Queue();
```

```
        MainQueue.push(15);
        MainQueue.push(16);
        MainQueue.push(17);
        MainQueue.push(14);
        MainQueue.pop();
        MainQueue.for_each();
        MainQueue.size();
        MainQueue.max_min();
        MainQueue.find(14);
        MainQueue.equals();
        MainQueue.get(2);
        MainQueue.empty();
        MainQueue.front();
        MainQueue.back();
        MainQueue.accumulate();


    }

}
```