МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Лабораторна робота № 1**

з дисципліни « ООП на Java »

*назва дисципліни*

на тему: «ПРОЕКТУВАННЯ КЛАСІВ ТА СТВОРЕННЯ ОБ'ЄКТІВ ЗАСОБАМИ МОВИ JAVA»

Виконав: студент 2 курсу групи № 622п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: доцент

Симбірський Генадій Дмитрович

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2023

ЗМІСТ

[**ПОСТАНОВКА ЗАВДАННЯ 3**](#_Toc148216875)

[**ЗМІСТ ЗВІТУ 3**](#_Toc148216876)

[**ПОРЯДОК ВИКОНАННЯ РОБОТИ 3**](#_Toc148216877)

[**ВИКОНАННЯ РОБОТИ 4**](#_Toc148216878)

[**ЗАГАЛЬНІ ВІДОМОСТІ 4**](#_Toc148216879)

[**ВИСНОВОК 11**](#_Toc148216880)

[**КОНТРОЛЬНІ ПИТАННЯ 12**](#_Toc148216881)

Мета роботи: отримання практичних навичок роботи з операторами мови програмування Java: оператори циклу та переходу.

ПОСТАНОВКА ЗАВДАННЯ

Навчитися працювати з операторами циклічних конструкцій while, do-while, for і конструкцій переходу break, continue, return.

# **ЗМІСТ ЗВІТУ**

1. Постановка завдання.
2. Встановити завдання, які вирішуються в лабораторній роботі
3. Коротко описати роботу оператора while.
4. Коротко описати роботу оператора do-while.
5. Коротко описати роботу оператора for.
6. Коротко описати роботу операторів переходу.
7. Записати код вирішення завдань.
8. Висновки по роботі.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Прочитати теоретичний матеріал
2. Вивчити роботу оператора while
3. Вивчити роботу оператора do-while
4. Вивчити роботу оператора for
5. Вивчити роботу конструкцій переходу
6. Зробити висновки по роботі.

# **ВИКОНАННЯ РОБОТИ**

## **Загальні відомості**

Мова програмування – Java

Операційна система – Windows 11

Процесор – Apple Silicon M1 Pro with 6 CPU cores

Тип компілятора – IntelliJ IDEA 2023

Оператор while

синтаксис циклу while наступний:

while(condition){

// action(s)

}

Приклад використання на рисунку 1

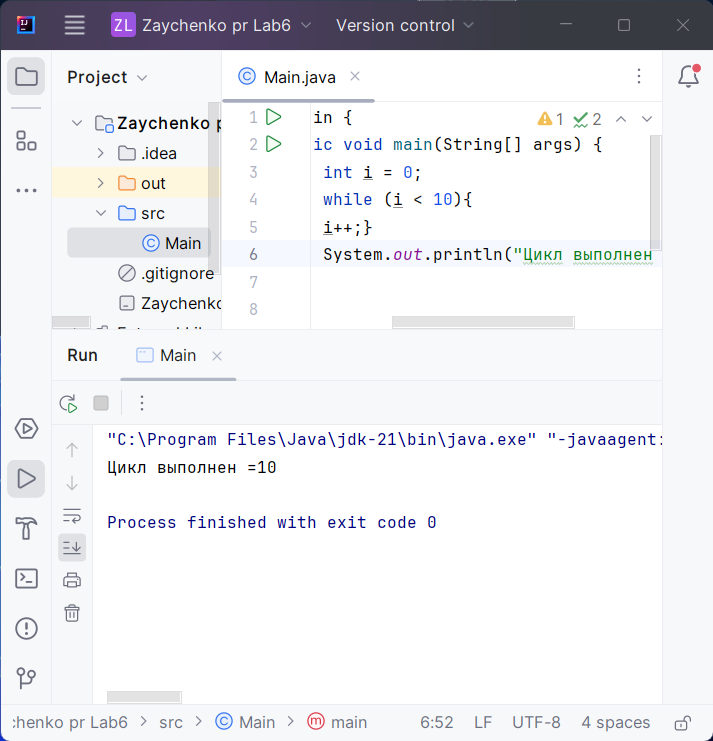


Рисунок 1 – використання циклу while

Оператор do-while

Форма конструкції do-while така:

do{

//action(s)

} while (condition)

Єдина відмінність циклу do-while від while полягає в тому, що цикл do- while виконується принаймні один раз, навіть якщо умова спочатку false. У циклі while, якщо умова спочатку false, тіло циклу ніколи не виконується. На практиці конструкція do-while вживається рідше, ніж while.

Приклад на рисунку 2 та 3.

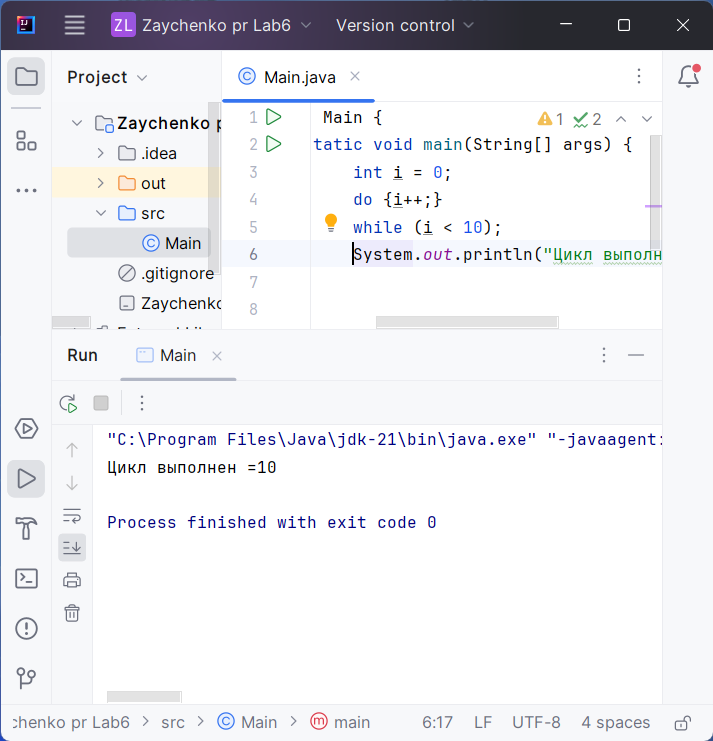


Рисунок 2 – багаторазове виконання циклу

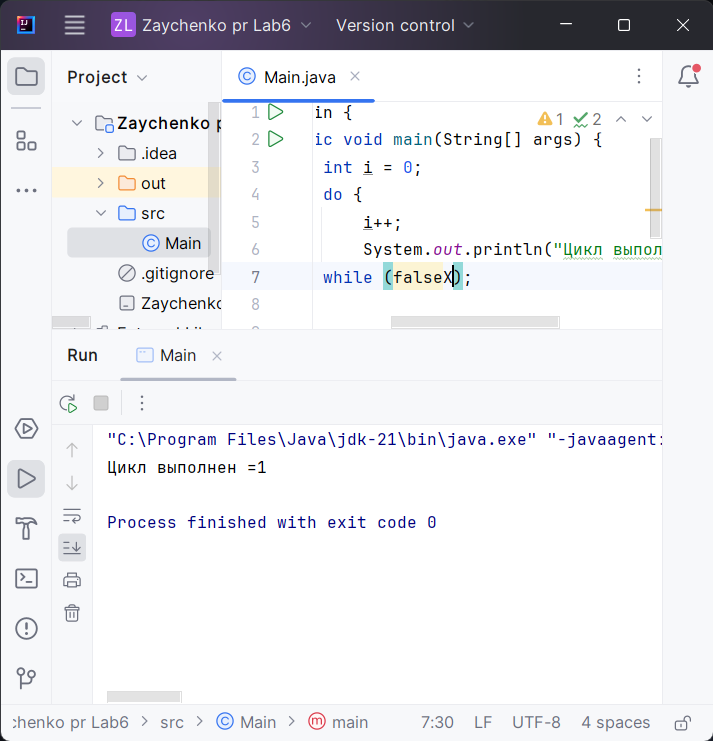


Рисунок 3 – одноразове виконання циклу

Оператор for

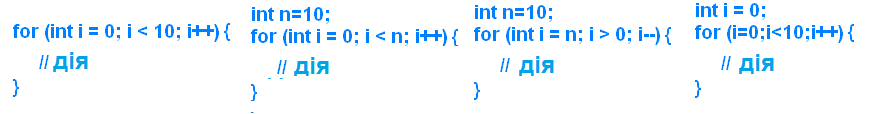
Конструкції for складають найбільш поширену різновид циклів. Цикл for проводить ініціалізацію перед першим кроком циклу. Потім виконується перевірка умови циклу, і в кінці кожної ітерації здійснюється якесь «прирощення» (зазвичай зміна керуючої змінної). Цикл for записується в такий спосіб:

for (initialization; condition; iteration) {

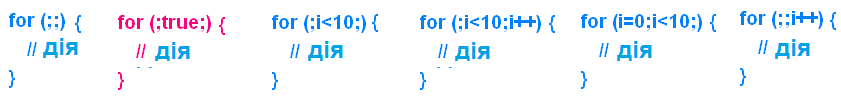
// code to be executed in each iteration

}

Приклади застосування



Приклади нескінченого циклу



Використання змінної у циклі. Результат на рисунку 4.

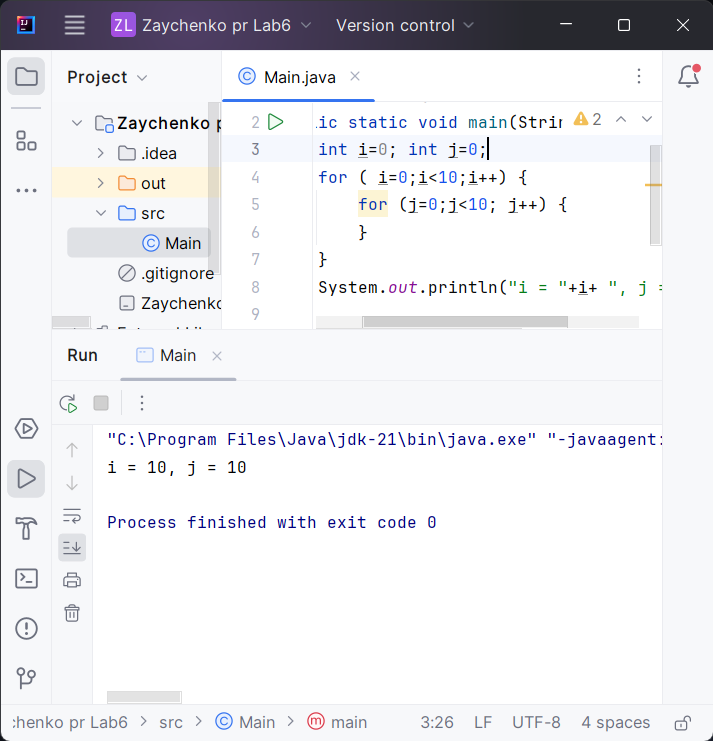


Рисунок 4 – Використання змінної у циклі

Знайдіть помилку

public static void main(String[] args) {

for (int i=0;i<10;i++) {

for (int j=0;3<10; j++) {

}

}

System.out.println("i = "+i+ ", j = "+j);

}

Використання змінних i та j поза областями видимості блоків for призводить до помилки компіляції.

Синтаксис foreach

for (element\_type variable : collection) {

// code that uses the variable

}

В Java SE5 з'явилася нова, більш компактна форма for для перебору елементів масивів і контейнерів. Ця спрощена форма, яка називається синтаксисом foreach, не вимагає ручного зміни службової змінної для перебору послідовності об'єктів - цикл автоматично представляє черговий елемент.

Для чого цикли?

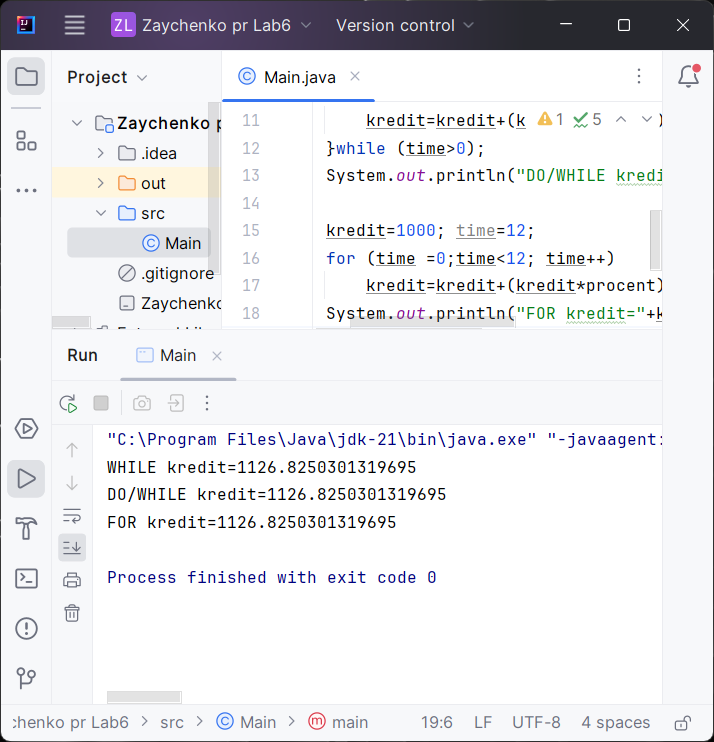


Рисунок 5 – приклад використання циклів

Конструкції переходу (return, break і continue)

У тілі будь-якого з циклів ви можете управляти потоком програми, використовуючи спеціальні ключові слова return, break і continue. Команда return завершує метод, а тому і цикл, команда break завершує цикл, при цьому оператори циклу, що залишилися не виконуються. Команда continue зупиняє виконання поточної ітерації циклу і переходить до початку циклу, щоб почати виконання нового кроку.

Наступні приклади програм показує приклад використання команд return (рис. 6), break (рис. 7) і continue (рис. 8) всередині циклів:

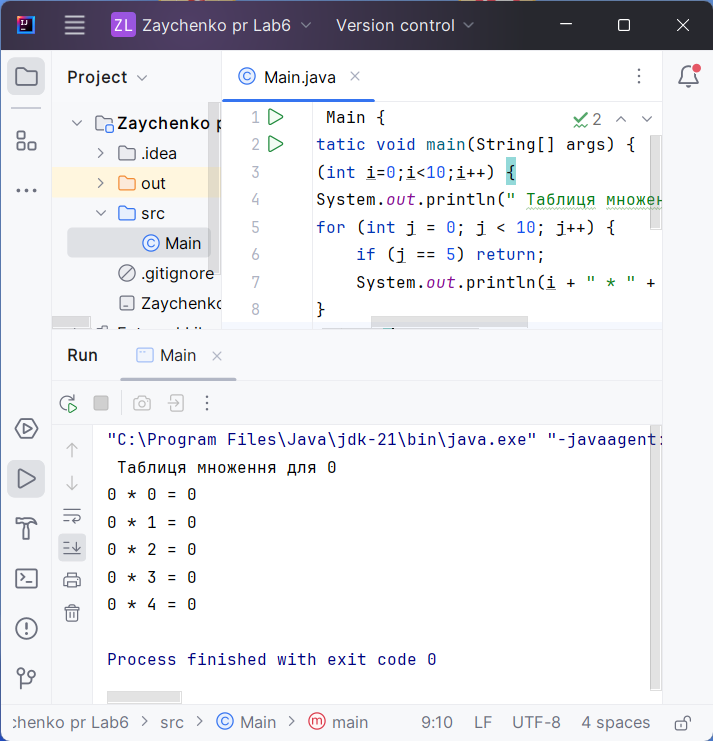


Рисунок 6 – оператор return

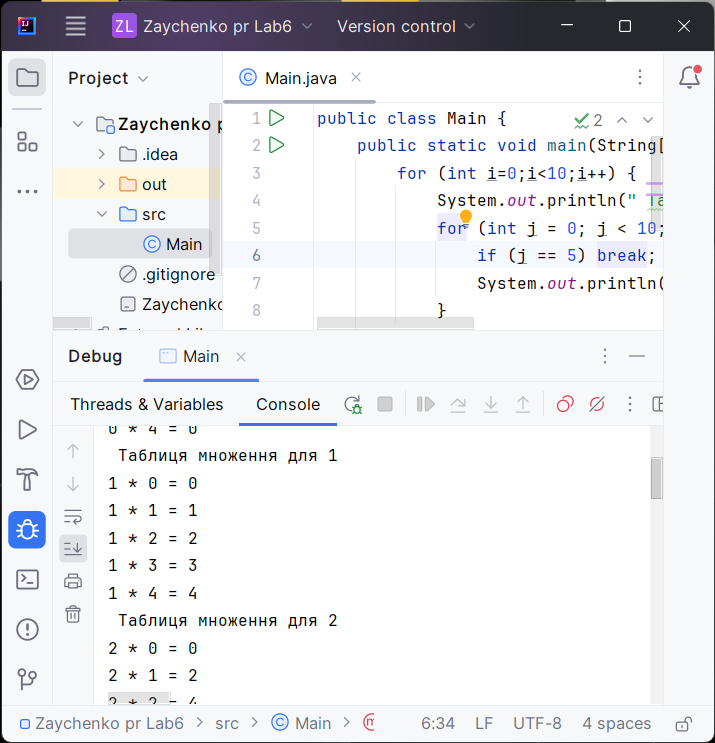


Рисунок 7 – оператор break

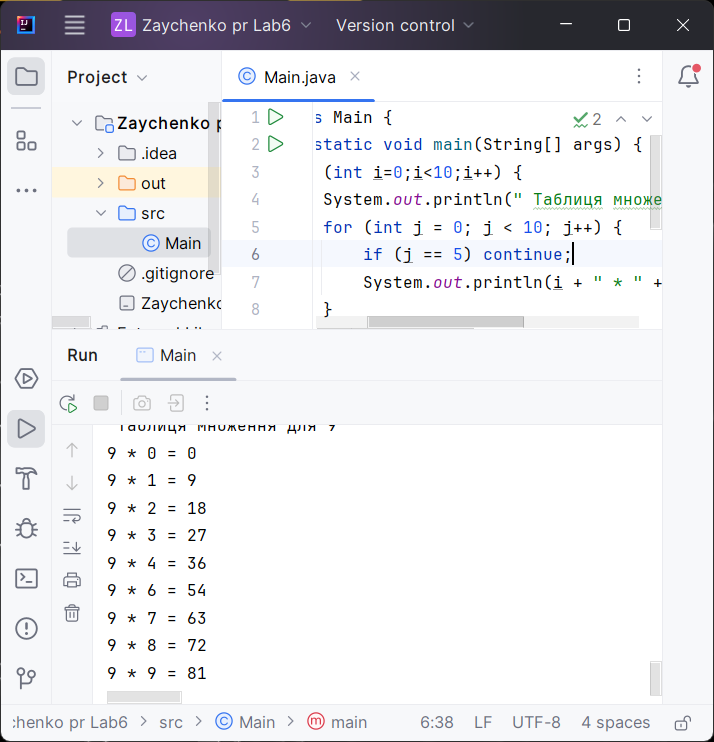


Рисунок 8 – оператор continue

Єдине місце, де в Java мітка може виявитися корисною, - прямо перед тілом циклу. Причому ніяких додаткових команд між міткою і тілом циклу бути не повинно. Причина приміщення мітки перед тілом циклу може бути лише одна - вкладення всередині циклу іншого циклу або конструкції вибору. Звичайні версії break і continue переривають тільки поточний цикл, в той час як їх версії з мітками здатні достроково завершувати цикли і передавати виконання в точку, що адресується міткою. Приклад на рисунку 9 та 10.

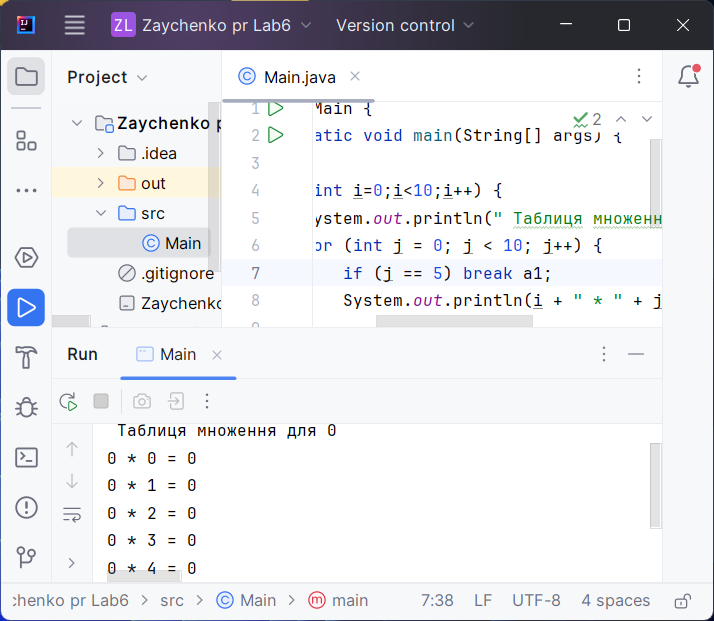


Рисунок 9 – цикл з оператором break a1

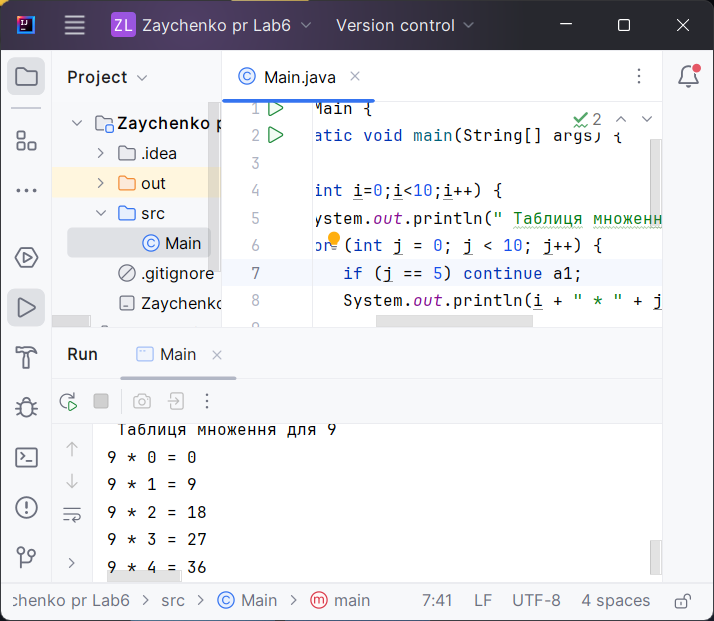


Рисунок 10 – цикл з оператором continue a1

Вирішення завдань

1. Вводиться натуральне число. Знайти суму парних цифр, що входять до його складу.

import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Введіть натуральне число: ");  
 long number = scanner.nextLong();  
 scanner.close();  
 int sumOfEvenDigits = calculateSumOfEvenDigits(number);  
 System.out.println("Сума парних цифр: " + sumOfEvenDigits);  
 }  
 private static int calculateSumOfEvenDigits(long num) {  
 int sum = 0;  
 String numStr = Long.toString(num);  
 for (int i = 0; i < numStr.length(); i++) {  
 int digit = Character.getNumericValue(numStr.charAt(i));  
 if (digit % 2 == 0) {  
 sum += digit;  
 }  
 }  
 return sum;  
 }  
}

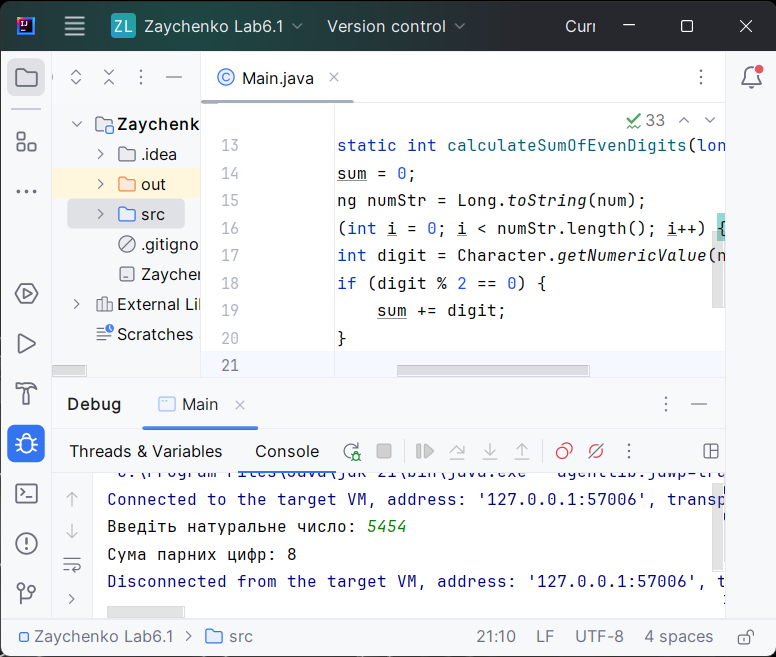


Рисунок 11 – результат роботи програми до завдання 1

1. Скласти таблицю значень функції y = 5 - x2/2 на відрізку [-5; 5] з кроком 0.5.

public class Main {  
 public static void main(String[] args) {  
 for (double x = -5.0; x <= 5.0; x += 0.5) {  
 double y = 5 - (x \* x / 2);  
 System.out.println("x= "+x+" y = "+y);  
 }  
 }  
}

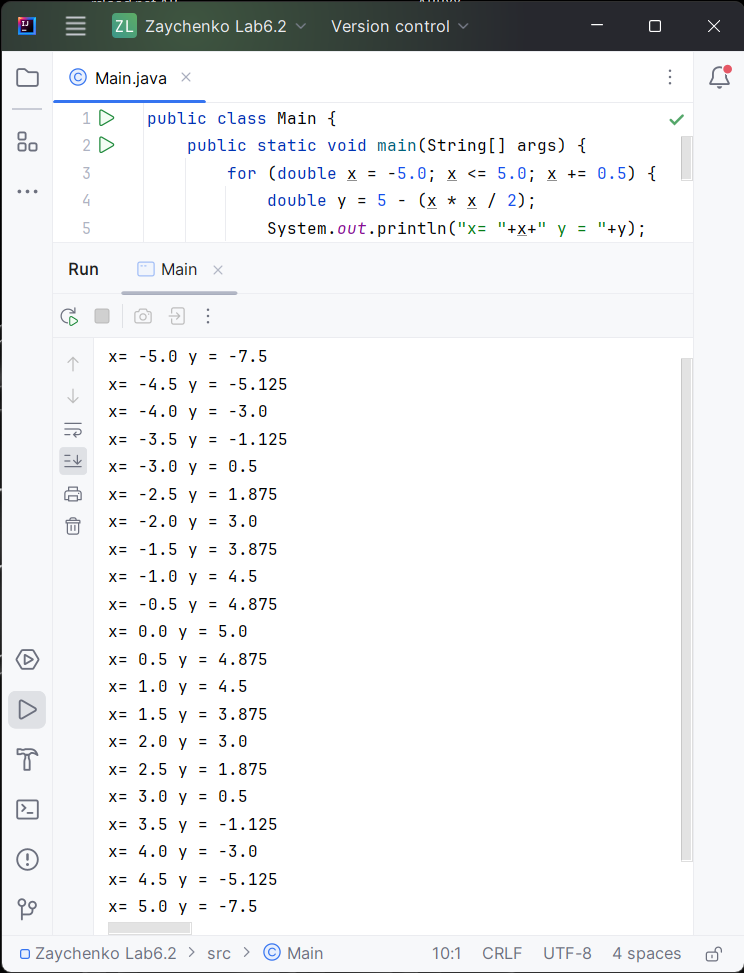


Рисунок 12 – результат роботи програми до завдання 2

1. Написати програму, яка підраховує кількість парних і непарних цифр в числі.

import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Введіть число: ");  
 long number = scanner.nextLong();  
 scanner.close();  
 int evenCount = countEvenDigits(number);  
 int oddCount = countOddDigits(number);  
 System.out.println("Кількість парних цифр: " + evenCount);  
 System.out.println("Кількість непарних цифр: " + oddCount);  
 }  
 private static int countEvenDigits(long num) {  
 int evenCount = 0;  
 String numStr = Long.toString(num);  
 for (int i = 0; i < numStr.length(); i++) {  
 int digit = Character.getNumericValue(numStr.charAt(i));  
 if (digit % 2 == 0)  
 evenCount++;  
 }  
 return evenCount;  
 }  
 private static int countOddDigits(long num) {  
 int oddCount = 0;  
 String numStr = Long.toString(num);  
 for (int i = 0; i < numStr.length(); i++) {  
 int digit = Character.getNumericValue(numStr.charAt(i));  
 if (digit % 2 != 0)  
 oddCount++;  
 }  
 return oddCount;  
 }  
}

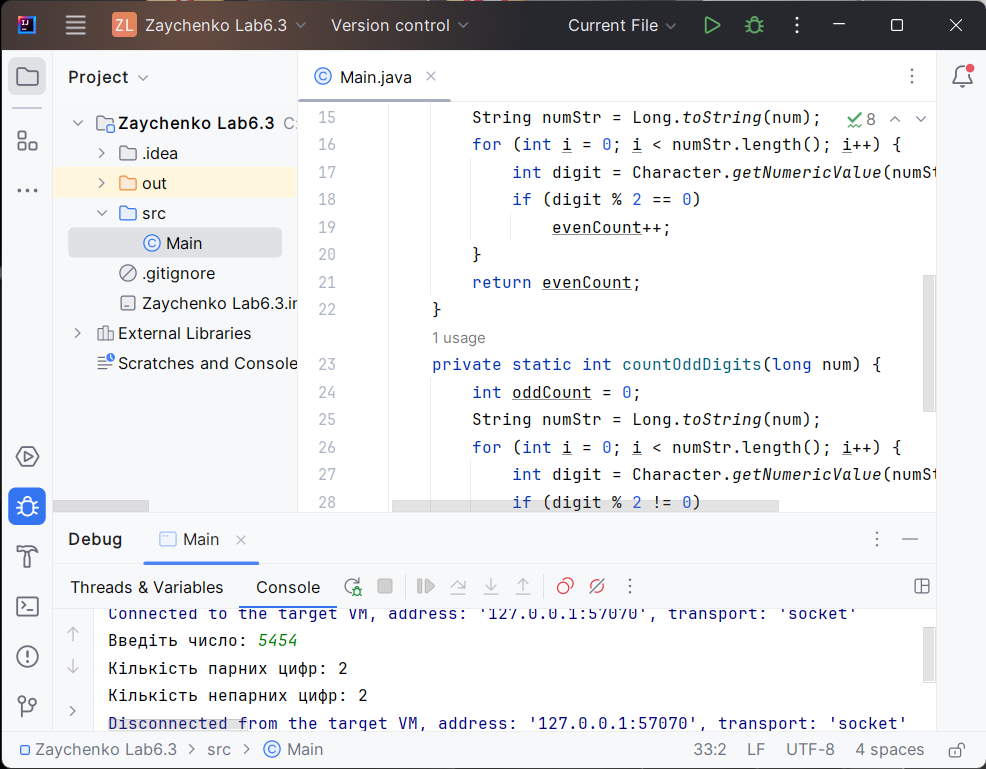


Рисунок 13 – результат роботи програми до завдання 3

1. Вводиться число. Перетворити його в інше число, цифри якого прямуватимуть в зворотному порядку в порівнянні з введенням числом.

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Введіть число: ");  
 long number = scanner.nextLong();  
 scanner.close();  
 long reversedNumber = reverseDigits(number);  
 System.out.println("Перетворене число: " + reversedNumber);  
 }  
 private static long reverseDigits(long num) {  
 long reversedNumber = 0;  
 while (num != 0) {  
 long digit = num % 10;  
 reversedNumber = reversedNumber \* 10 + digit;  
 num /= 10;  
 }  
 return reversedNumber;  
 }  
}

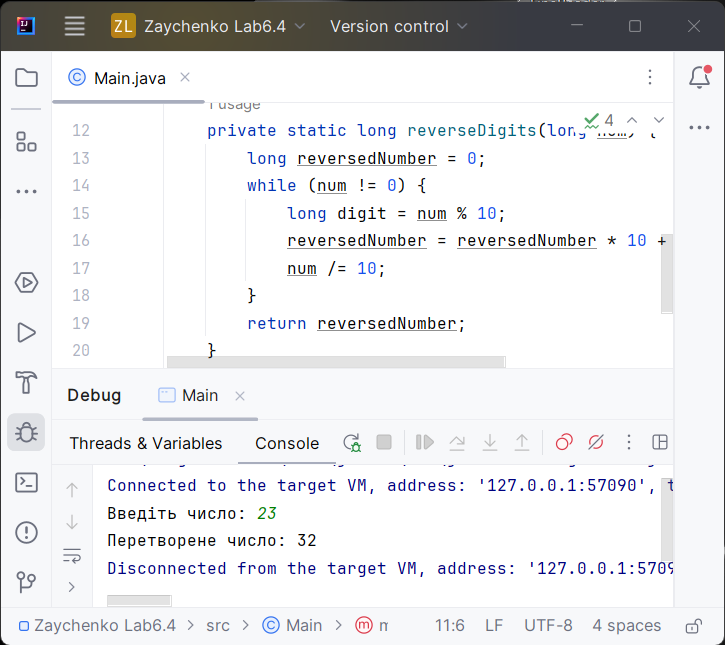


Рисунок 14 – результат роботи програми до завдання 4

1. Вивести на екран куби чисел від A до B, які вводить користувач.

import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Введіть число A: ");  
 int A = scanner.nextInt();  
 System.out.print("Введіть число B: ");  
 int B = scanner.nextInt();  
 scanner.close();  
 displayCubesInRange(A, B);  
 }  
 private static void displayCubesInRange(int start, int end) {  
 System.out.println("Куби чисел від " + start + " до " + end + ":");  
 for (int i = start; i <= end; i++) {  
 int cube = i \* i \* i;  
 System.out.println("Куб " + i + " = " + cube);  
 }  
 }  
}

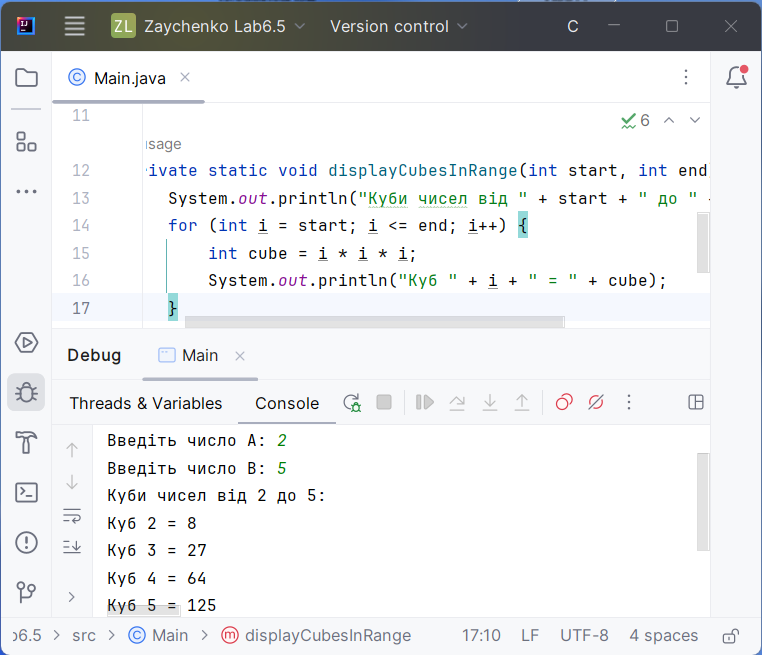


Рисунок 15 – результат роботи програми до завдання 5

1. Обчислити факторіал числа. Факторіал числа є твір всіх натуральних чисел від 1 до цього числа включно.

import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Введіть натуральне число: ");  
 int number = scanner.nextInt();  
 scanner.close();  
 long factorial = calculateFactorial(number);  
 System.out.println("Факторіал числа " + number + " = " + factorial);  
 }  
 private static long calculateFactorial(int num) {  
 if (num < 0) {  
 System.out.println("Введено від'ємне число. Факторіал визначений тільки для невід'ємних чисел.");  
 System.exit(0);}  
 if (num == 0 || num == 1)  
 return 1;  
 long factorial = 1;  
 for (int i = 2; i <= num; i++)  
 factorial \*= i;  
 return factorial;  
 }  
}

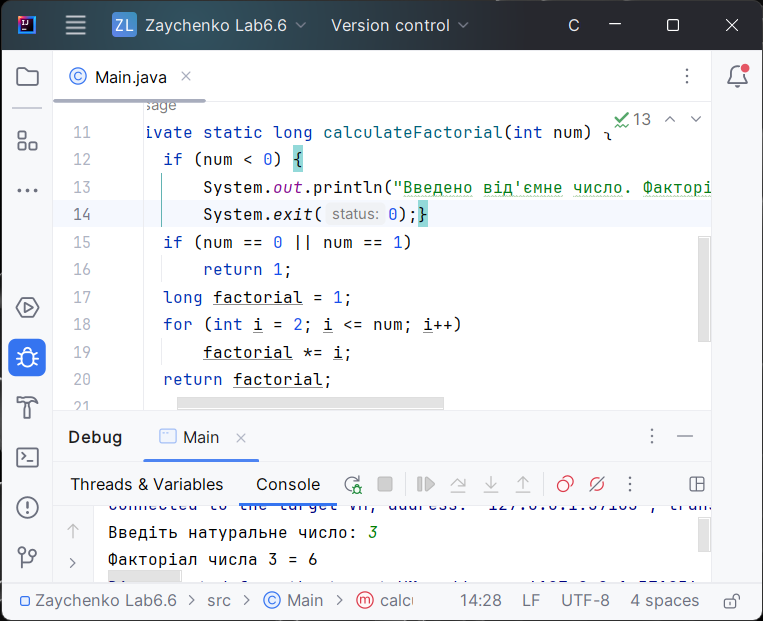


Рисунок 16 – результат роботи програми до завдання 6

1. Вводяться цілі числа до першого числа, яке менше двох. Визначити, скільки простих чисел було введено.

import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 int primeCount = 0;  
 System.out.println("Введіть цілі числа (введення припиниться, якщо число менше двох):");  
 while (true) {  
 int number = scanner.nextInt();  
 if (number < 2)  
 break;  
 if (Simple(number))  
 primeCount++;  
 }  
 scanner.close();  
 System.out.println("Кількість простих чисел: " + primeCount);  
 }  
  
 private static boolean Simple(int num) {  
 if (num <= 1) {  
 return false;  
 } else {  
 for (int i = 2; i <= Math.sqrt(num); i++)  
 if (num % i == 0)  
 return false;  
 return true;  
 }  
 }  
}

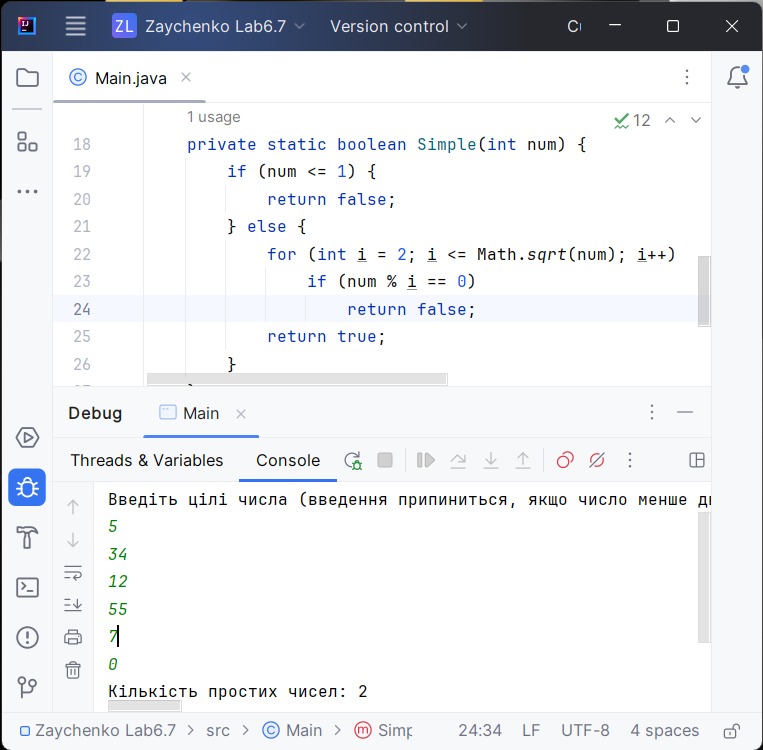


Рисунок 17 – результат роботи програми до завдання 7

1. Визначити з яких цифр складається число

import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Введіть ціле число: ");  
 long number = scanner.nextLong();  
 scanner.close();  
 System.out.print("Цифри, з яких складається число " + number + ": ");  
 separate\_digit(number);  
 System.out.println("");  
 }  
 private static void separate\_digit(long num) {  
 if (num == 0)  
 return;  
 separate\_digit(num / 10);  
 System.out.print(num % 10 + " ");  
 }  
}

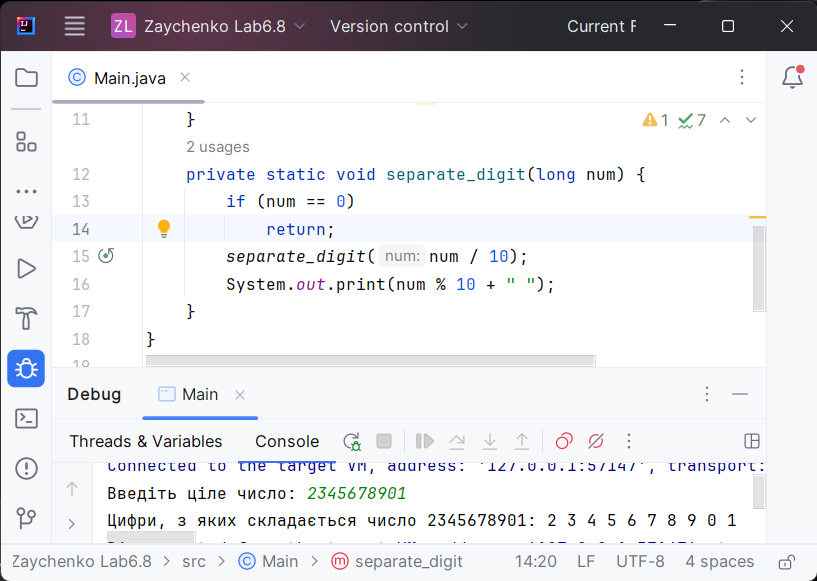


Рисунок 18 – результат роботи програми до завдання 8

1. Дано число. Знайти суму і твір його цифр.

import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Введіть ціле число: ");  
 long number = scanner.nextLong();  
 scanner.close();  
 int sum = sumOfDigits(number);  
 long multiply = multiplyOfDigits(number);  
 System.out.println("Сума цифр числа: " + sum);  
 System.out.println("Добуток цифр числа: " + multiply);  
 }  
 private static int sumOfDigits(long num) {  
 int sum = 0;  
 while (num != 0) {  
 sum += num % 10;  
 num /= 10;  
 }  
 return sum;  
 }  
 private static long multiplyOfDigits(long num) {  
 long mult = 1;  
 while (num != 0) {  
 mult \*= num % 10;  
 num /= 10;  
 }  
 return mult;  
 }  
}

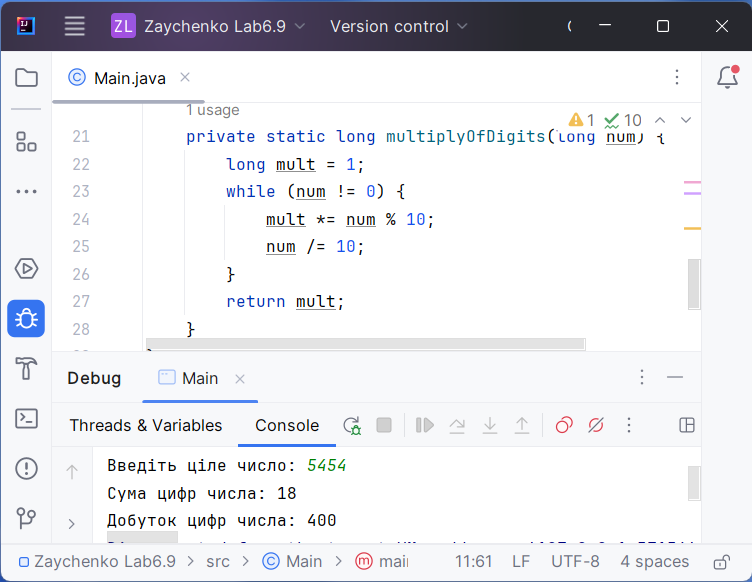


Рисунок 19 – результат роботи програми до завдання 9

1. Вивести на екран ряд чисел Фібоначчі, що складається з n елементів.

import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Введіть кількість елементів ряду Фібоначчі: ");  
 int n = scanner.nextInt();  
 scanner.close();  
 System.out.println("Ряд Фібоначчі з " + n + " елементів:");  
 fibonacci(n);  
 }  
 private static void fibonacci(int n) {  
 int first = 0, second = 1;  
 for (int i = 0; i < n; i++) {  
 System.out.print(first + " ");  
 int next = first + second;  
 first = second;  
 second = next;  
 }  
 }  
}

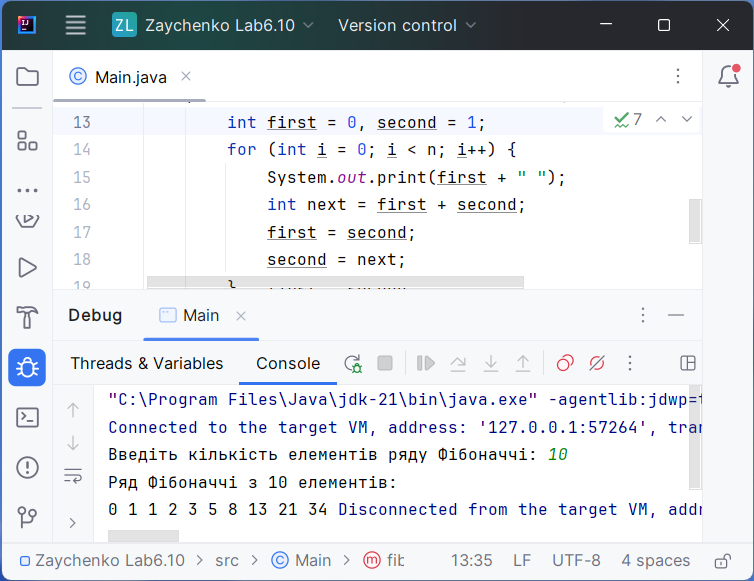


Рисунок 20 – результат роботи програми до завдання 10

# **ВИСНОВОК**

Отримано практичні навички роботи з операторами мови програмування Java: оператори циклу і переходу.

# **Контрольні питання**

1. **Особливості роботи while?**

* while - це цикл, який виконується, доки вказана умова є true.
* Перед кожним виконанням тіла циклу перевіряється умова. Якщо вона виконується, виконується тіло циклу.

1. **Особливості роботи do-while?**

* do-while - це цикл, який виконується принаймні один раз, і потім повторюється, доки вказана умова є true.
* Умова перевіряється після виконання тіла циклу, тому тіло гарантовано виконується принаймні один раз.

1. **Особливості роботи for?**

* for - це цикл, який складається з ініціалізації, умови та виразу ітерації.
* Ініціалізація виконується один раз перед початком циклу. Умова перевіряється перед кожним виконанням тіла циклу. Вираз ітерації виконується після кожного виконання тіла циклу.

1. **Особливості роботи break?**

* break використовується для негайного виходу з циклу (for, while, do-while) або switch.
* При виклику break виконання поточного циклу або блоку switch припиняється, і програма виходить з нього.

1. **Особливості роботи continue?**

* continue використовується для переходу до наступної ітерації циклу.
* Коли викликається continue, будь-які залишкові інструкції в тілі циклу пропускаються, і виконується наступна ітерація циклу.

1. **Особливості роботи return?**

* return використовується для повернення значення з методу (функції) та завершення його виконання.
* При виклику return виконання поточного методу припиняється, і контроль передається викликаючому коду, а передане значення (якщо воно є) стає результатом виклику методу.