**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**Кафедра інтелектуальних технологій**

**ЛАБОРАТОРНА РОБОТА №4**

з дисципліни «Об'єктно-орієнтоване програмування»

Тема роботи: «Інтерфейси та внутрішні класи Java»

Виконала студентка

групи АнД-21

Радоманова С.П.

Перевірила:

Москаленко Н.В.

**Київ-2024**

Мета роботи є закріплення знань та набуття практичних навичок:

− створення ієрархії інтерфейсів java;

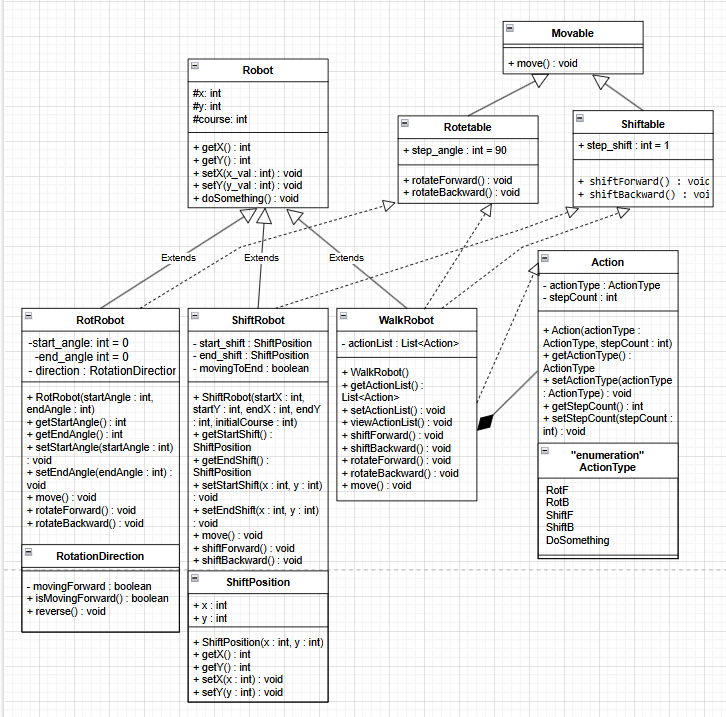
− створення ієрархії класів java, що реалізують описані інтерфейси;

− створення та використання внутрішніх, вкладених та анонімних класів java;

− використання пакетів java;

− використання перелічень при описі класів.

Модифікована UML-діаграми класів з описом нових доданих класів:



## **Опис структури класів**

### **Клас: Robot**

**Призначення:** Описує базовий функціонал робота, включаючи координати та напрямок руху.

**Атрибути:**

int x — координата робота по осі X.

int y — координата робота по осі Y.

int course — курс (напрямок руху) робота.

**Методи:**

getX(): Повертає значення координати X.

getY(): Повертає значення координати Y.

setX(int x\_val): Встановлює значення координати X.

setY(int y\_val): Встановлює значення координати Y.

doSomething(): Виконує певну місію та виводить повідомлення у консоль: *"Робот виконав місію!"*.

## **Дочірні класи робота:**

### **Клас: RotRobot**

**Призначення:** Розширює функціональність базового робота, додаючи можливість обертатися.

**Батьківський клас:** Robot

**Реалізує інтерфейс:** Rotetable

**Атрибути:**

int start\_angle — початковий кут.

int end\_angle — кінцевий кут.

RotationDirection direction — екземпляр внутрішнього класу для визначення напряму обертання.

**Внутрішній клас:**

RotationDirection — відповідає за логіку напряму обертання.

Атрибути:

boolean movingForward — напрямок обертання (true — вперед, false — назад).

Методи:

isMovingForward(): Повертає поточний напрямок обертання.

reverse(): Змінює напрямок обертання на протилежний.

**Методи:**

**Конструктор:** Ініціалізує об'єкт початковим (start\_angle) і кінцевим (end\_angle) кутами. Встановлює початковий курс (course) рівним start\_angle.

getStartAngle(): Повертає початковий кут.

getEndAngle(): Повертає кінцевий кут.

setStartAngle(int startAngle): Встановлює початковий кут.

setEndAngle(int endAngle): Встановлює кінцевий кут.

move(): Виконує обертання вперед або назад залежно від поточного напряму. Змінює напрямок обертання, якщо досягнуто межі (початкового або кінцевого кута).

**Методи інтерфейсу Rotetable:**

rotateForward(): Обертає робота вперед на кут step\_angle, не перевищуючи end\_angle.

rotateBackward(): Обертає робота назад на кут step\_angle, не виходячи за межу start\_angle.

doSomething(): Викликається після кожного обертання, повідомляючи про виконання дії.

### **Клас: ShiftRobot**

**Призначення:** Розширює базовий клас Robot, додаючи можливість зміщуватися між двома позиціями в просторі.

**Батьківський клас:** Robot

**Реалізує інтерфейси:** Shiftable, Movable

**Атрибути:**

ShiftPosition start\_shift — початкова позиція (координати x і y).

ShiftPosition end\_shift — кінцева позиція.

boolean movingToEnd — визначає напрямок руху (до кінцевої позиції чи до початкової).

**Вкладений клас: ShiftPosition**

**Призначення:** Представляє координати позиції.

**Атрибути:**

int x — координата x.

int y — координата y.

**Методи:**

Гетери та сетери для координат x і y.

**Методи:**

**Конструктор:** Приймає початкові координати (startX, startY), кінцеві координати (endX, endY) та початковий напрямок (initialCourse). Ініціалізує позиції start\_shift та end\_shift.

getStartShift(), getEndShift(): Повертають початкову та кінцеву позиції.

setStartShift(int x, int y), setEndShift(int x, int y): Встановлюють координати початкової та кінцевої позицій.

move(): Реалізує рух між початковою та кінцевою позиціями, змінюючи напрямок при досягненні меж.

**Методи інтерфейсу Shiftable:**

shiftForward(): Зміщення вперед залежно від орієнтації (course).

shiftBackward(): Зміщення назад залежно від орієнтації (course).

### **Клас: WalkRobot**

**Призначення:** Реалізує комплексну поведінку робота, який може рухатися, повертатися та виконувати список дій.

**Батьківський клас:** Robot

**Реалізує інтерфейси:** Movable, Rotetable, Shiftable

**Атрибути:**

List<Action> actionList — список дій, які виконує робот.

**Методи:**

**Конструктор:** Створює порожній список дій actionList.

getActionList(): Повертає список дій.

setActionList(): Дозволяє користувачеві додати дії до списку через консоль. Користувач вибирає тип дії (ShiftF, ShiftB, RotF, RotB, DoSomething) та кількість повторень.

viewActionList(): Виводить список дій із зазначенням типу та кількості кроків.

**Методи інтерфейсу Shiftable:**

shiftForward(): Зміщення вперед залежно від напряму.

shiftBackward(): Зміщення назад залежно від напряму.

**Методи інтерфейсу Rotetable:**

rotateForward(): Обертання за годинниковою стрілкою на кут step\_angle.

rotateBackward(): Обертання проти годинникової стрілки на кут step\_angle.

**Метод інтерфейсу Movable:**

move(): Виконує всі дії зі списку actionList послідовно, відповідно до їх типу та кількості кроків.Після кожної дії виводить поточну позицію (x, y) та напрямок (course).

### **Клас: Action**

**Призначення:** Описує дію, яку може виконувати робот.

**Атрибути:**

ActionType actionType — тип дії (наприклад, поворот, рух вперед, виконання завдання тощо).

int stepCount — кількість кроків, які необхідно виконати для даної дії.

**Методи:**

**Конструктор:** Ініціалізує об'єкт класу з переданими значеннями типу дії та кількості кроків.

getActionType(): Повертає тип дії.

setActionType(ActionType actionType): Встановлює новий тип дії.

getStepCount(): Повертає кількість кроків для дії.

setStepCount(int stepCount): Встановлює кількість кроків.

**Вкладений клас:**

static enum ActionType — перелік можливих типів дій:

RotF — поворот вперед.

RotB — поворот назад.

ShiftF — зміщення вперед.

ShiftB — зміщення назад.

DoSomething — виконання певного завдання.

## **Інтерфейси:**

### **Інтерфейс: Movable**

**Призначення:** Визначає обов'язковий метод для об'єктів, які можуть рухатися.

**Методи:**

move(): Абстрактний метод, який реалізується в класах, що підтримують рух.

### **Інтерфейс: Rotetable**

**Призначення:** Визначає поведінку об'єктів, які можуть обертатися.

**Константи:**

int step\_angle = 90 — кут обертання за один крок (read-only за замовчуванням).

**Методи:**

rotateForward(): Метод для обертання вперед на заданий кут.

rotateBackward(): Метод для обертання назад на заданий кут.

### 

### **Інтерфейс: Shiftable**

**Призначення:** Визначає поведінку об'єктів, які можуть зміщуватися в просторі.

**Константи:**

int step\_shift = 1 — кількість одиниць зміщення за один крок (read-only за замовчуванням).

**Методи:**

shiftForward(): Метод для зміщення вперед на задану кількість одиниць.

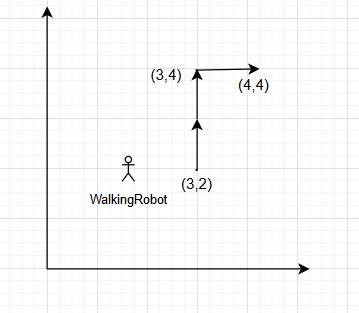
shiftBackward(): Метод для зміщення назад на задану кількість одиниць.

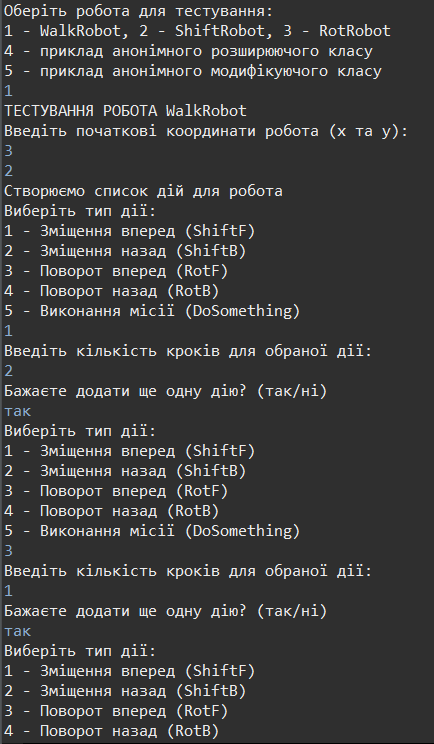
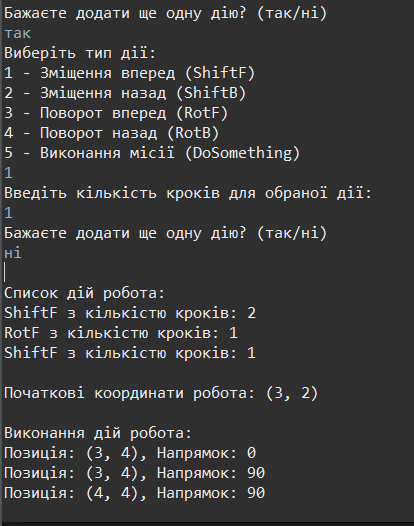
# Тестування:

Спочатку створимо робота WalkRobot.

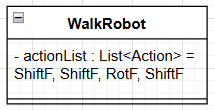
Робот робить зміщення вперед (2 кроки), робить один поворот вперед (по осі х в додатному напрямку = вперед) і знову робить зміщення вперед (1 крок).

Схематичне зображення:

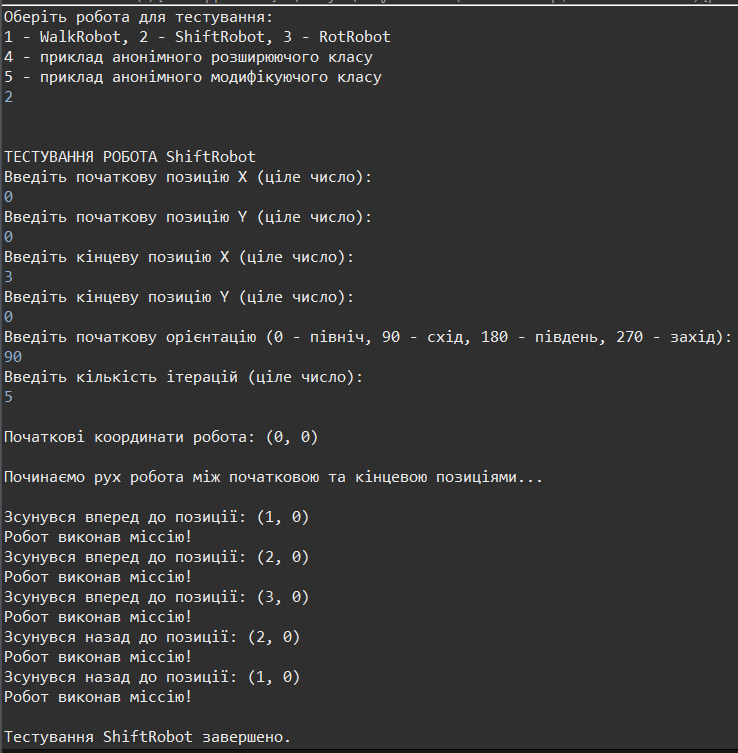




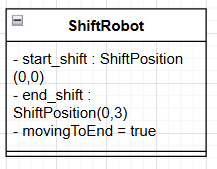
UML діаграма об’єкту:



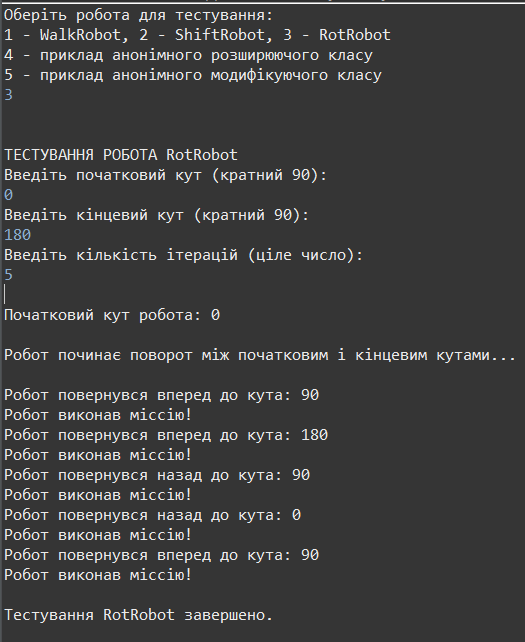
Тепер проведемо тестування ShiftRobot. Робот починає з координат (0, 0) та йде на схід до точки (0, 3). Оскільки у нас зазначено 5 ітерацій, то по доходженні заданої точки він розвертається в протилежний бік і йде назад.



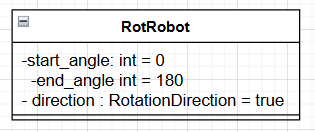
UML діаграма об’єкту:



Тепер проведемо тестування RotRobot. Робот починає з позиції 0, після чого здійснює 5 ітерацій в бік 180 градусів (90 => 180 => 90 => 0 => 90, доходить до 180 - змінює напрямок на протилежний, доходить до 0 - змінює напрямок на протилежний)



UML діаграма об’єкту:



Текст програми:

main:

package robots1;

import java.util.Scanner;

public class main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Оберіть робота для тестування: ");

System.***out***.println("1 - WalkRobot, 2 - ShiftRobot, 3 - RotRobot");

System.***out***.println("4 - приклад анонімного розширюючого класу");

System.***out***.println("5 - приклад анонімного модифікуючого класу");

int option = scanner.nextInt();

switch (option){

case 1:

// ТЕСТУВАННЯ РОБОТА WalkRobot

System.***out***.println("ТЕСТУВАННЯ РОБОТА WalkRobot");

// Створення екземпляра WalkRobot

WalkRobot robot = new WalkRobot();

System.***out***.println("Введіть початкові координати робота (x та y): ");

robot.setX(scanner.nextInt());

robot.setY(scanner.nextInt());

// Заповнення списку дій користувачем

robot.setActionList();

// Перегляд списку дій

System.***out***.println("\nСписок дій робота:");

robot.viewActionList();

System.***out***.println("\nПочаткові координати робота: (" + robot.getX() + ", " + robot.getY() + ")");

// Виконання руху робота згідно з діями

System.***out***.println("\nВиконання дій робота:");

robot.move();

break;

case 2:

// ТЕСТУВАННЯ РОБОТА ShiftRobot

System.***out***.println("\n\nТЕСТУВАННЯ РОБОТА ShiftRobot");

// Отримуємо початкові значення від користувача

System.***out***.println("Введіть початкову позицію X (ціле число): ");

int startX = scanner.nextInt();

System.***out***.println("Введіть початкову позицію Y (ціле число): ");

int startY = scanner.nextInt();

System.***out***.println("Введіть кінцеву позицію X (ціле число): ");

int endX = scanner.nextInt();

System.***out***.println("Введіть кінцеву позицію Y (ціле число): ");

int endY = scanner.nextInt();

System.***out***.println("Введіть початкову орієнтацію (0 - північ, 90 - схід, 180 - південь, 270 - захід): ");

int initialCourse = scanner.nextInt();

System.***out***.println("Введіть кількість ітерацій (ціле число): ");

int iterations = scanner.nextInt();

// Створення екземпляра ShiftRobot із введеними параметрами

ShiftRobot robot2 = new ShiftRobot(startX, startY, endX, endY, initialCourse);

System.***out***.println("\nПочаткові координати робота: (" + robot2.getX() + ", " + robot2.getY() + ")");

System.***out***.println("\nПочинаємо рух робота між початковою та кінцевою позиціями...\n");

// Імітуємо кілька кроків руху

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

robot2.move();

}

System.***out***.println("\nТестування ShiftRobot завершено.");

break;

case 3:

// ТЕСТУВАННЯ РОБОТА RotRobot

System.***out***.println("\n\nТЕСТУВАННЯ РОБОТА RotRobot");

// Введення початкових значень від користувача

System.***out***.println("Введіть початковий кут (кратний 90): ");

int startAngle = scanner.nextInt();

System.***out***.println("Введіть кінцевий кут (кратний 90): ");

int endAngle = scanner.nextInt();

System.***out***.println("Введіть кількість ітерацій (ціле число): ");

int iterations2 = scanner.nextInt();

// Створення екземпляра RotRobot із введеними параметрами

RotRobot robot3 = new RotRobot(startAngle, endAngle);

System.***out***.println("\nПочатковий кут робота: " + robot3.getStartAngle());

System.***out***.println("\nРобот починає поворот між початковим і кінцевим кутами...\n");

// Імітуємо кілька кроків повороту

for (int i = 0; i < iterations2; i++) {

robot3.move();

}

System.***out***.println("\nТестування RotRobot завершено.");

break;

case 4:

// ТЕСТУВАННЯ АНОНІМНОГО КЛАСУ З РОЗШИРЕННЯМ

// НА ПРИКЛАДІ Robot

Robot robot4 = new Robot();

System.***out***.println("Введіть початкові координати робота (x та y): ");

robot4.setX(scanner.nextInt());

robot4.setY(scanner.nextInt());

System.***out***.println("Robot виконує метод doSomething: ");

robot4.doSomething();

Robot robot5 = new Robot(){ //Розширення базового методу doSomething

public void doSomething(){

System.***out***.println("Робот виконав місію за допомогою розширеного базового методу!");

}

};

System.***out***.println("Введіть початкові координати робота (x та y): ");

robot5.setX(scanner.nextInt());

robot5.setY(scanner.nextInt());

System.***out***.println("Robot виконує метод doSomething: ");

robot5.doSomething();

break;

case 5:

// ТЕСТУВАННЯ МОДИФІКУЮЧОГО АНОНІМНОГО КЛАСУ

// НА ПРИКЛАДІ ІНТЕРФЕЙСУ Movable

Movable robot6 = new Movable() {

*@Override*

public void move() {

System.***out***.println("Метод move було викликано!");

}

};

System.***out***.println("Тестування методу move для екземпляра інтерфейсу Movable ");

robot6.move();

default:

System.***out***.println("\nВведено неправильне число");

break;

}

}

}

Movable:

package robots1;

public interface Movable {

void move();

}

Rotateable:

package robots1;

public interface Rotetable {

int ***step\_angle*** = 90; // readOnly за замовчуванням

void rotateForward();

void rotateBackward();

}

Shiftable:

package robots1;

public interface Shiftable {

int ***step\_shift*** = 1; // readOnly за замовчуванням

void shiftForward();

void shiftBackward();

}

Action:

package robots1;

public class Action {

private *ActionType* actionType; // Тип дії

private int stepCount; // Кількість кроків для даної дії

// Конструктор

public Action(*ActionType* actionType, int stepCount) {

this.actionType = actionType;

this.stepCount = stepCount;

}

// Метод для отримання типу дії

public *ActionType* getActionType() {

return actionType;

}

// Метод для встановлення типу дії

public void setActionType(*ActionType* actionType) {

this.actionType = actionType;

}

// Метод для отримання кількості кроків

public int getStepCount() {

return stepCount;

}

// Метод для встановлення кількості кроків

public void setStepCount(int stepCount) {

this.stepCount = stepCount;

}

public static enum *ActionType* {

***RotF***, ***RotB***, ***ShiftF***, ***ShiftB***, ***DoSomething***

}

}

RotRobot:

package robots1;

public class RotRobot extends Robot implements Rotetable {

private int start\_angle = 0;

private int end\_angle = 0;

private RotationDirection direction; // внутрішній клас для напрямку обертання

// Внутрішній клас для відстеження напрямку обертання (завдання 4)

private class RotationDirection {

private boolean movingForward = true; // Логічна змінна для напряму

public boolean isMovingForward() {

return movingForward;

}

public void reverse() {

movingForward = !movingForward; // Змінює напрямок на протилежний

}

}

// Конструктор з параметрами

public RotRobot(int startAngle, int endAngle) {

this.start\_angle = startAngle;

this.end\_angle = endAngle;

this.course = startAngle; // Ініціалізуємо course початковим кутом

this.direction = new RotationDirection(); // Створюємо екземпляр внутрішнього класу

}

// Гетери

public int getStartAngle() {

return start\_angle;

}

public int getEndAngle() {

return end\_angle;

}

// Сетери

public void setStartAngle(int startAngle) {

this.start\_angle = startAngle;

}

public void setEndAngle(int endAngle) {

this.end\_angle = endAngle;

}

public void move() {

if (direction.isMovingForward()) { // Перевіряємо напрямок обертання

rotateForward();

if (course >= end\_angle) {

direction.reverse(); // Змінюємо напрямок, якщо досягли межі

}

} else {

rotateBackward();

if (course <= start\_angle) {

direction.reverse(); // Змінюємо напрямок, якщо досягли початку

}

}

}

*@Override*

public void rotateForward() {

if (course + ***step\_angle*** <= end\_angle) {

course += ***step\_angle***;

} else {

course = end\_angle; // course не може перевищувати end\_angle

}

System.***out***.println("Робот повернувся вперед до кута: " + course);

doSomething();

}

*@Override*

public void rotateBackward() {

if (course - ***step\_angle*** >= start\_angle) {

course -= ***step\_angle***;

} else {

course = start\_angle; // course не може бути меншим за start\_angle

}

System.***out***.println("Робот повернувся назад до кута: " + course);

doSomething();

}

}

ShiftRobot:

package robots1;

public class ShiftRobot extends Robot implements Shiftable, Movable {

// Вкладений клас для представлення позицій з координатами x та y(завдання 4)

protected static class ShiftPosition {

int x;

int y;

public ShiftPosition(int x, int y) {

this.x = x;

this.y = y;

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public void setX(int x) {

this.x = x;

}

public void setY(int y) {

this.y = y;

}

}

protected ShiftPosition start\_shift; // Початкова позиція

protected ShiftPosition end\_shift; // Кінцева позиція

protected boolean movingToEnd = true; // Напрямок руху: true - до кінцевої позиції, false - до початкової

// Конструктор

public ShiftRobot(int startX, int startY, int endX, int endY, int initialCourse) {

this.start\_shift = new ShiftPosition(startX, startY);

this.end\_shift = new ShiftPosition(endX, endY);

this.course = initialCourse;

}

// Гетери і сетери для start\_shift і end\_shift

public ShiftPosition getStartShift() {

return start\_shift;

}

public ShiftPosition getEndShift() {

return end\_shift;

}

public void setStartShift(int x, int y) {

this.start\_shift.setX(x);

this.start\_shift.setY(y);

}

public void setEndShift(int x, int y) {

this.end\_shift.setX(x);

this.end\_shift.setY(y);

}

*@Override*

public void move() {

// Рух між початковою та кінцевою позиціями

if (movingToEnd) {

shiftForward(); // Рух вперед

if (this.x == end\_shift.getX() && this.y == end\_shift.getY()) {

movingToEnd = false; // Зміна напрямку на зворотний

}

} else {

shiftBackward(); // Рух назад

if (this.x == start\_shift.getX() && this.y == start\_shift.getY()) {

movingToEnd = true; // Зміна напрямку на вперед

}

}

}

*@Override*

public void shiftForward() {

// Рух у напрямку поточної орієнтації

switch (course) {

case 0: // Північ

y += ***step\_shift***;

break;

case 90: // Схід

x += ***step\_shift***;

break;

case 180: // Південь

y -= ***step\_shift***;

break;

case 270: // Захід

x -= ***step\_shift***;

break;

default:

System.***out***.println("Неправильна орієнтація. Орієнтація повинна бути кратною 90 градусам.");

return;

}

System.***out***.println("Зсунувся вперед до позиції: (" + x + ", " + y + ")");

doSomething();

}

*@Override*

public void shiftBackward() {

// Рух у протилежному напрямку поточної орієнтації

switch (course) {

case 0: // Південь

y -= ***step\_shift***;

break;

case 90: // Захід

x -= ***step\_shift***;

break;

case 180: // Північ

y += ***step\_shift***;

break;

case 270: // Схід

x += ***step\_shift***;

break;

default:

System.***out***.println("Неправильна орієнтація. Орієнтація повинна бути кратною 90 градусам.");

return;

}

System.***out***.println("Зсунувся назад до позиції: (" + x + ", " + y + ")");

doSomething();

}

}

WalkRobot:

package robots1;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class WalkRobot extends Robot implements Movable, Rotetable, Shiftable {

private List<Action> actionList;

// Конструктор за замовчуванням

public WalkRobot() {

this.actionList = new ArrayList<>();

}

// Гетер для списку дій

public List<Action> getActionList() {

return actionList;

}

// Сетер для списку дій, який заповнюється користувачем

public void setActionList() {

Scanner scanner = new Scanner(System.in);

List<Action> actions = new ArrayList<>();

boolean addMoreActions = true;

System.out.println("Створюємо список дій для робота");

// Додавання дій користувачем

while (addMoreActions) {

System.out.println("Виберіть тип дії:");

System.out.println("1 - Зміщення вперед (ShiftF)");

System.out.println("2 - Зміщення назад (ShiftB)");

System.out.println("3 - Поворот вперед (RotF)");

System.out.println("4 - Поворот назад (RotB)");

System.out.println("5 - Виконання місії (DoSomething)");

int actionChoice = scanner.nextInt();

Action.ActionType actionType = null;

switch (actionChoice) {

case 1 -> actionType = Action.ActionType.ShiftF;

case 2 -> actionType = Action.ActionType.ShiftB;

case 3 -> actionType = Action.ActionType.RotF;

case 4 -> actionType = Action.ActionType.RotB;

case 5 -> actionType = Action.ActionType.DoSomething;

default -> {

System.out.println("Неправильний вибір. Спробуйте ще раз.");

continue;

}

}

System.out.println("Введіть кількість кроків для обраної дії:");

int stepCount = scanner.nextInt();

// Додавання дії до списку

actions.add(new Action(actionType, stepCount));

// Запит на продовження додавання дій

System.out.println("Бажаєте додати ще одну дію? (так/ні)");

String userResponse = scanner.next();

if (!userResponse.equalsIgnoreCase("так")) {

addMoreActions = false;

}

}

this.actionList = actions; // Присвоєння списку дій об'єкту робота

scanner.close();

}

// Метод для перегляду списку дій

public void viewActionList() {

for (Action action : actionList) {

System.out.println(action.getActionType() + " з кількістю кроків: " + action.getStepCount());

}

}

@Override

public void shiftForward() {

switch (course) {

case 0 -> y += step\_shift;

case 90 -> x += step\_shift;

case 180 -> y -= step\_shift;

case 270 -> x -= step\_shift;

}

}

@Override

public void shiftBackward() {

switch (course) {

case 0 -> y -= step\_shift;

case 90 -> x -= step\_shift;

case 180 -> y += step\_shift;

case 270 -> x += step\_shift;

}

}

@Override

public void rotateForward() {

course = (course + step\_angle) % 360;

}

@Override

public void rotateBackward() {

course = (course - step\_angle + 360) % 360;

}

@Override

public void move() {

for (Action action : actionList) {

for (int i = 0; i < action.getStepCount(); i++) {

switch (action.getActionType()) {

case RotF -> rotateForward();

case RotB -> rotateBackward();

case ShiftF -> shiftForward();

case ShiftB -> shiftBackward();

case DoSomething -> doSomething();

}

}

// Вивід позиції після кожної дії

System.out.println("Позиція: (" + x + ", " + y + "), Напрямок: " + course);

}

}

}