МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»

КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

**ІНФОРМАЦІЙНА СИСТЕМА**

**«ZERNO»**

Курсовий проєкт з дисципліни

«Об’єктно-орієнтоване програмування»

Виконали

Здобувачі ВО групи ПІ-221 Автомієнко Я. Г., Гречко І. О.

Керівник

Старший викладач Бивойно Т. П.

Чернігів, 2024 рік

**ТЕХНІЧНЕ ЗАВДАННЯ**

на виконання курсового проєкту з дисципліни " Об’єктно-орієнтоване програмування"

здобувачів ВО Автомієнко Я. Г., Гречко І. О., група ПІ-221

**Тема роботи:** Інформаційна система "Zerno"

**Очікувані технічні та експлуатаційні результати роботи:**

Java проєкт з графічним інтерфейсом, який створено у середовищі розробки IntelliJ IDEA, який забезпечує роботу з інформаційною системою «Zerno» (комбайн, трактор, приймальний пункт).

На початку програми трактори чекають у черзі до комбайнів. Як тільки комбайн заповниться, трактор під’їзжає його розвантажити. Після розвантаження комбайн продовжує працювати, а трактор відправляється до приймальних пунктів. Трактор стає у чергу до приймальних пунктів. Якщо є вільні приймальні пункти, вони починають розвантажувати трактори. Після розвантаження приймальних пункт може почати розвантажувати інший трактор, а розвантажені трактори їдуть назад до комбайнів, де знову встають у чергу.

* Дана інформація реалізовується у вигляді організації взаємодії об’єктів, що працюють паралельно.

**Планова трудомісткість роботи:** 4 кредити **(**120 годин).

|  |  |  |
| --- | --- | --- |
| Виконавці роботи: |  | Автомієнко Я. Г., Гречко І.О. |
|  |  |  |
| Керівник роботи: |  | Бивойно Т. П. |

ЗМІСТ

[ВСТУП 4](#_Toc166757625)

[ПОСТАНОВКА ЗАДАЧІ 4](#_Toc166757626)

[ОГЛЯД МЕТОДУ РЕАЛІЗАЦІЇ 5](#_Toc166757627)

[РЕАЛІЗАЦІЯ ПРОЄКТУ 8](#_Toc166757629)

[ТЕСТУВАННЯ ПРОЄКТУ 43](#_Toc166757630)

[ВИСНОВКИ 46](#_Toc166757631)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 47](#_Toc166757632)

# ВСТУП

Будь-яка людина потребує своєчасного доступу до інформації. Цінність інформації в сучасному світі дуже висока. Роль розпорядників інформації найчастіше виконують інформаційно-довідкові системи. Такі системи надають своєчасний доступ до інформації і можливість безпечно зберігати її в структурованому вигляді. Практично будь-яка сучасна організація потребує інформаційну систему, що задовольняє ті чи інші потреби по зберіганню, управлінню та адмініструванню даних.

На сьогоднішній день на ринку представлено безліч технологій доступу до даних і серверів інформаційних систем, кожна з яких має свої відмінні риси. Сучасні програми обробки даних орієнтовані на роботу з великою кількістю користувачів, на їх віддаленість від місця розташування основного сервера БД.

Завданням курсового проєкту є реалізація інформаційної системи з графічним інтерфейсом у вигляді організації взаємодії об’єктів, що працюють паралельно. При створенні програми було використано середовище розробки IntelliJ IDEA і мову програмування Java.

Об’єкт дослідження – інформаційно-довідкова система «Zerno».

Мета роботи – розробка інформаційної системи відповідно до варіанту, яка містить інформацію про комбайни, трактори, приймальні пункти та кількість розвантаженого зерна. Дану систему необхідно представити у організації взаємодії об’єктів, що працюють паралельно.

# ПОСТАНОВКА ЗАДАЧІ

Необхідно реалізувати інформаційну систему з графічним інтерфейсом, яка міститиме наступну інформацію про комбайни, трактори, приймальні пункти та кількість розвантаженого зерна.

Система повинна відповідати таким вимогам:

* надійність;
* безпека;
* інформація, що зберігається в системі, повинна бути захищена від аварійних ситуацій.

Інформаційна система «Zerno» міститиме список комбайнів, тракторів та приймальних пунктів, які працюють паралельно.

# ОГЛЯД МЕТОДУ РЕАЛІЗАЦІЇ

Дане завдання передбачає створення програми, основне завдання якої буде зберігання даних про комбайни, трактори, приймальні пункти та кількість розвантаженого зерна, а також роботу з цими даними.

Поставлене завдання можна реалізувати у вигляді організації взаємодії об’єктів, що працюють паралельно.

Дану структуру та її відображення можемо реалізувати за допомогою бібліотеки JavaFX. За допомогою класу Thread будується відповідна модель задачі та робота з нею.

## Структура даних інформаційної системи «Zerno»

Темою курсового проєкту було створення інформаційної системи «Zerno».

На рисунку 2.1 наведено діаграму класів інформаційної системи «Zerno».

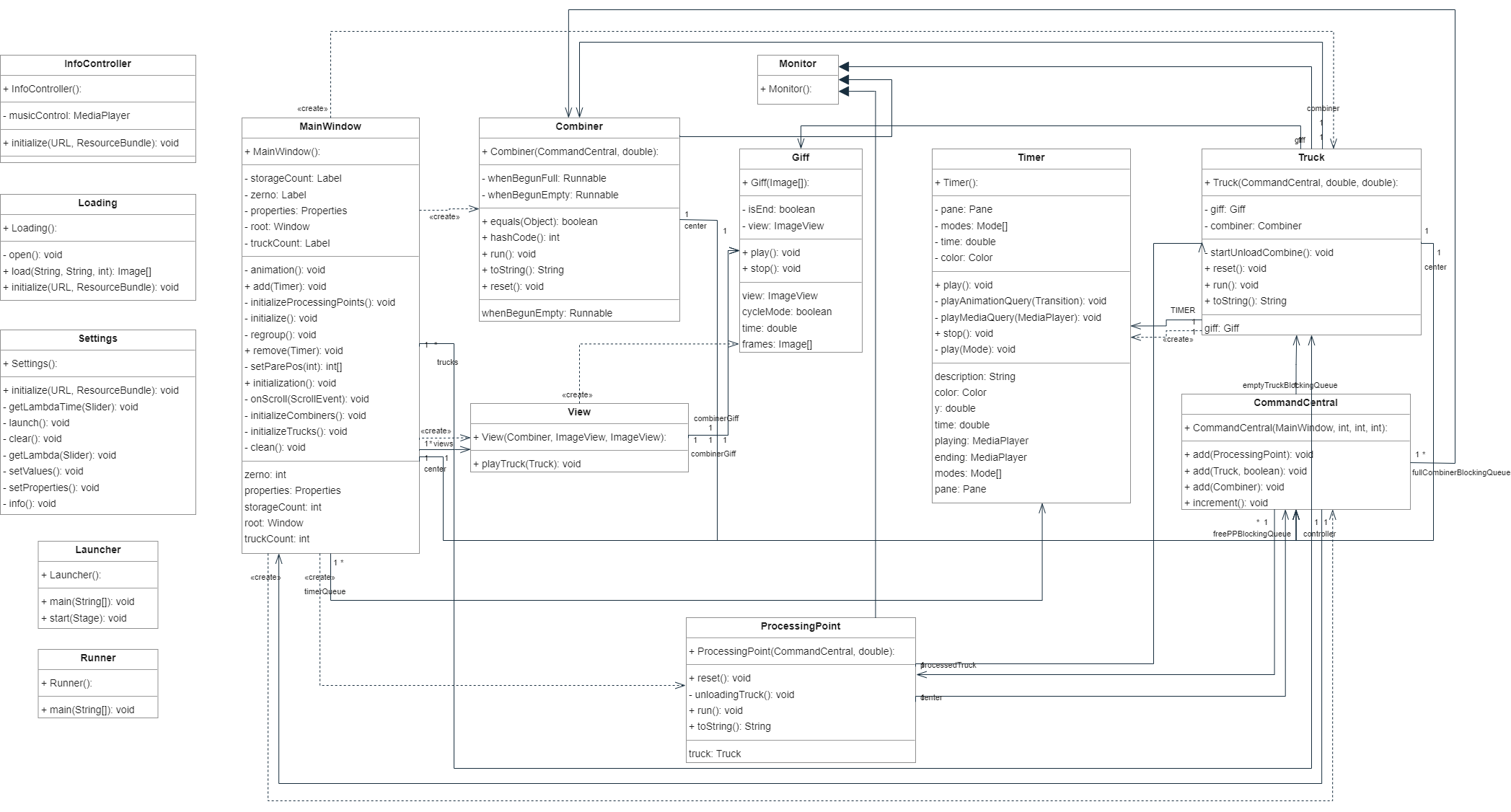


Рисунок 2.1 – Діаграма класів інформаційної системи «Zerno»

На рисунку 2.2 зображено ієрархію класів інформаційної системи «Zerno»

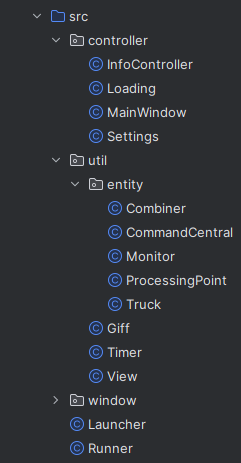


Рисунок 2.2 – Ієрархія класів інформаційної системи «Zerno»

Проєкт містить декілька пакетів з ресурсами:

* Пакет «controller» містить класи, які відповідають за логіку графічних вікон програми
* Пакет «util» містить допоміжні класи для роботи с моделлю даних інформаційної системи «Zerno»
* Пакет «util.entity» містить класи-моделі інформаційної системи «Zerno»
* Пакет «window» містить стилі, які відображають графічні вікна на екрані.

# РЕАЛІЗАЦІЯ ПРОЄКТУ

В лістингу 4.1 наведено код класу Launcher.

Лістинг 4.1 – Код класу Launcher.

|  |
| --- |
| import javafx.application.Application; import javafx.application.Platform; import javafx.fxml.FXMLLoader; import javafx.scene.Parent; import javafx.scene.Scene; import javafx.stage.Stage;  import java.util.Objects;  public class Launcher extends Application {   @Override  public void start(Stage primaryStage) throws Exception {  Parent root = FXMLLoader.*load*(  Objects.*requireNonNull*(getClass().getResource("window/LoadWindow.fxml"))  );  primaryStage.setOnCloseRequest(event -> {  System.*exit*(0);  });  primaryStage.setScene(new Scene(root, 325, 50));  primaryStage.setTitle("Завантаження...");  primaryStage.setResizable(false);  primaryStage.show();  }   public static void main(String[] args) {  *launch*(args);  }  } |

В лістингу 4.2 наведено код класу Loading.

Лістинг 4.2 – Код класу Loading.

|  |
| --- |
| pa package controller;  import javafx.application.Platform; import javafx.fxml.FXML; import javafx.fxml.FXMLLoader; import javafx.fxml.Initializable; import javafx.scene.Parent; import javafx.scene.Scene; import javafx.scene.control.ProgressBar; import javafx.scene.image.Image; import javafx.stage.Stage; import javafx.stage.StageStyle;  import java.io.IOException; import java.net.URL; import java.util.Objects; import java.util.ResourceBundle; import java.util.concurrent.atomic.AtomicInteger;  public class Loading implements Initializable {   @FXML  private ProgressBar bar;   public static Image[] *COMBINER\_FRAMES*;  public static Image[] *COMBINER\_LIGHT\_BULB\_FRAMES*;  public static Image[] *TRUCK\_FRAMES*;   private final Thread imageLoader = new Thread(() -> {  *COMBINER\_FRAMES* = load("/combiner/nonLightBulb/", "comb", 25);  *COMBINER\_LIGHT\_BULB\_FRAMES* = load("/combiner/lightBulb/", "comb", 25);  *TRUCK\_FRAMES* = load("/truck/", "truck", 194);  Platform.*runLater*(this::open);  });   @Override  public void initialize(URL location, ResourceBundle resources) {  imageLoader.start();  }   private void open() {  ((Stage) bar.getScene().getWindow()).close();  Stage primaryStage = new Stage();  Settings.*mainStage* = primaryStage;  primaryStage.initStyle(StageStyle.*UNDECORATED*);  primaryStage.setOnCloseRequest(windowEvent -> System.*exit*(0));  Parent root;  try {  root = FXMLLoader.*load*(Objects.*requireNonNull*(getClass().getResource("/window/Settings.fxml")));  } catch (IOException e) {  throw new RuntimeException(e);  }  Scene scene = new Scene(root, 600, 425);  scene.getStylesheets().add("style/style.css");  primaryStage.setScene(scene);  primaryStage.setTitle("Налаштування");  primaryStage.setResizable(false);  primaryStage.getIcons().add(new Image(Objects.*requireNonNull*(getClass().getResource("/icon.png")).toString()));  primaryStage.show();  }   private final AtomicInteger all = new AtomicInteger();   public Image[] load(String url, String name, int size) {   Image[] frames = new Image[size];  for (int i = 0; i < size; i++) {  frames[i] = new Image(  Objects.*requireNonNull*(getClass().getResource(url + name + i + ".png")).toString()  //"file: " + url + name + i + ".png"  );  all.incrementAndGet();  bar.setProgress(all.get() / 244D);  }  return frames;  } } |

В лістингу 4.3 наведено код класу Settings.

Лістинг 4.3 – Код класу Settings.

|  |
| --- |
| package controller;  import javafx.animation.FadeTransition; import javafx.fxml.FXML; import javafx.fxml.FXMLLoader; import javafx.fxml.Initializable; import javafx.scene.Parent; import javafx.scene.Scene; import javafx.scene.control.Button; import javafx.scene.control.Label; import javafx.scene.control.Slider; import javafx.scene.image.Image; import javafx.scene.layout.Pane; import javafx.scene.media.Media; import javafx.scene.media.MediaPlayer; import javafx.stage.Modality; import javafx.stage.Stage; import javafx.stage.StageStyle; import javafx.stage.WindowEvent; import javafx.util.Duration;  import java.io.IOException; import java.net.URL; import java.util.Objects; import java.util.Properties; import java.util.ResourceBundle;  public class Settings implements Initializable {   @FXML  private Slider PPsCount;   @FXML  private Slider combinersCount;   @FXML  private Slider goingFromCombToPPTime;   @FXML  private Slider goingFromPPToCombTime;   @FXML  private Slider processingTime;   @FXML  private Slider trucksCount;   @FXML  private Slider harvestTimeSlider;   @FXML  private Label lbl;   @FXML  private Button cls;   public static Stage mainStage;   @FXML  private Pane top;  private double deltaX, deltaY;   private FadeTransition fd;   private final Media music = new Media(  Objects.requireNonNull(getClass().getResource("/mainTheme.mp3")).toString()  );  private final MediaPlayer musicControl = new MediaPlayer(music);   private final Properties properties = new Properties();   private double harvestTime;  private int combinerCount;  private double toPPTime;  private double toCombinersTime;  private int truckCount;  private double receptionTime;  private int PPCount;   @Override  public void initialize(URL url, ResourceBundle resourceBundle) {   musicControl.setOnEndOfMedia(() -> {  musicControl.seek(musicControl.getStartTime());  musicControl.play();  });  musicControl.play();   fd = new FadeTransition(Duration.seconds(0.3), lbl);  fd.setFromValue(1D);  fd.setToValue(0D);   getLambda(combinersCount);  getLambda(PPsCount);  getLambda(trucksCount);   getLambdaTime(harvestTimeSlider);  getLambdaTime(goingFromCombToPPTime);  getLambdaTime(goingFromPPToCombTime);  getLambdaTime(processingTime);   InfoController.setMusicControl(musicControl);   top.setOnMousePressed(event -> {  deltaX = event.getSceneX();  deltaY = event.getSceneY();  });   top.setOnMouseDragged(event -> {  mainStage.setX(event.getScreenX() - deltaX);  mainStage.setY(event.getScreenY() - deltaY);  });   cls.setOnAction(event -> mainStage.fireEvent(new WindowEvent(mainStage, WindowEvent.WINDOW\_CLOSE\_REQUEST)));  }   @FXML  private void launch() throws Exception {  clear();   setValues();  setProperties();   Stage mainStage = new Stage();  MainWindow.*mainStage* = mainStage;  mainStage.initStyle(StageStyle.*UNDECORATED*);  FXMLLoader loader = new FXMLLoader(  getClass().getResource("/window/MainWindow.fxml")  );  Parent root = loader.load();   MainWindow window = loader.getController();  window.setRoot(mainStage);  window.setProperties(properties);  window.initialization();   Scene scene = new Scene(root, 1280, 725);  scene.getStylesheets().add("style/style.css");  mainStage.setScene(scene);  mainStage.setTitle("Zerno");  mainStage.getIcons().add(new Image(Objects.*requireNonNull*(getClass().getResource("/icon.png")).toString()));  mainStage.setResizable(false);  mainStage.initModality(Modality.*APPLICATION\_MODAL*);  mainStage.show();  }   @FXML  private void info() throws IOException {  if (InfoController.mainStage == null) {  Stage mainStage = new Stage();  InfoController.mainStage = mainStage;  mainStage.initStyle(StageStyle.UNDECORATED);  Parent root = FXMLLoader.load(  Objects.requireNonNull(getClass().getResource("/window/Info.fxml"))  );  Scene scene = new Scene(root, 550, 335);  scene.getStylesheets().add("/style/style.css");  mainStage.setScene(scene);  mainStage.setTitle("Info");  mainStage.getIcons().add(new Image(Objects.requireNonNull(getClass().getResource("/icon.png")).toString()));  mainStage.setResizable(false);  mainStage.show();  } else  InfoController.mainStage.toFront();  }   private void clear() {  for (Thread t : Thread.getAllStackTraces().keySet())  if (t.getName().equals("JFXMedia Player EventQueueThread"))  t.interrupt();  }   private void setValues() {  harvestTime = harvestTimeSlider.getValue() / 2;  combinerCount = (int) combinersCount.getValue();  toPPTime = goingFromCombToPPTime.getValue() / 2;  toCombinersTime = goingFromPPToCombTime.getValue() / 2;  truckCount = (int) trucksCount.getValue();  receptionTime = processingTime.getValue() / 2;  PPCount = (int) PPsCount.getValue();  }   private void setProperties() {  properties.put("harvestTime", harvestTime);  properties.put("combinerCount", combinerCount);  properties.put("toPPTime", toPPTime);  properties.put("toCombinersTime", toCombinersTime);  properties.put("truckCount", truckCount);  properties.put("receptionTime", receptionTime);  properties.put("PPCount", PPCount);  }   private void getLambda(Slider slider) {  slider.setOnMousePressed(event -> {  fd.stop();  lbl.setOpacity(1.0);  lbl.setLayoutY(slider.getLayoutY());  lbl.setLayoutX(slider.getLayoutX() + slider.getWidth() + 5);  lbl.setText(String.valueOf((int) slider.getValue()));  });   slider.setOnMouseReleased(event -> {  fd.stop();  fd.playFromStart();  });   slider.valueProperty().addListener((obs, oldVal, newVal) -> {  int snappedValue = (int) Math.round(newVal.doubleValue());  slider.setValue(snappedValue);  lbl.setText(String.valueOf(snappedValue));   });  }   private void getLambdaTime(Slider slider) {  slider.setOnMousePressed(event -> {  fd.stop();  lbl.setOpacity(1.0);  lbl.setLayoutY(slider.getLayoutY());  lbl.setLayoutX(slider.getLayoutX() + slider.getWidth() + 5);  lbl.setText(slider.getValue() / 2D + " c.");  });    slider.setOnMouseReleased(event -> {  fd.stop();  fd.playFromStart();  });   slider.valueProperty().addListener((obs, oldVal, newVal) -> {  int snappedValue = (int) Math.round(newVal.doubleValue());  slider.setValue(snappedValue);  lbl.setText(snappedValue / 2D + " c.");  });  }  } |

В лістингу 4.4 наведено код класу InfoController.

Лістинг 4.4 – Код класу InfoController.

|  |
| --- |
| package controller;  import javafx.fxml.FXML; import javafx.fxml.Initializable; import javafx.scene.control.Button; import javafx.scene.control.Label; import javafx.scene.control.Slider; import javafx.scene.layout.Pane; import javafx.scene.media.MediaPlayer; import javafx.stage.Stage; import javafx.stage.WindowEvent;  import java.net.URL; import java.util.ResourceBundle;  public class InfoController implements Initializable {   @FXML  private Label variant;   @FXML  private Label task;   @FXML  private Label dev;   @FXML  private Slider volumeSlider;   @FXML  private Pane sound;   @FXML  private Button cls;   @FXML  private Pane topInfo;  private double deltaX, deltaY;   private static MediaPlayer *musicControl*;  public static Stage *mainStage*;   public static void setMusicControl(MediaPlayer musicControl) {  InfoController.*musicControl* = musicControl;  musicControl.setVolume(0.3);  }   @Override  public void initialize(URL url, ResourceBundle resourceBundle) {  *mainStage*.setOnCloseRequest(event -> *mainStage* = null);   sound.setStyle("-fx-background-image: url('settings/sound.png');");   variant.setText("""  Моделювання зернозбиральних робіт  Вараіант: 8 - Zerno  """);  task.setText("Сільськогосподарський загін по збиранню зерна складається з декількох зернозбиральних " +  "комбайнів й автомобілів для вивезення зерна з поля. Комбайни працюють цілодобово. " +  "Коли комбайн намолотить повний бункер зерна, він зупиняється й чекає автомобіля, " +  "якщо його ще немає. Коли автомобіль під'їжджає, зерно перевантажується в кузов автомобіля, " +  "і комбайн продовжує роботу. Автомобіль відвозить зерно на елеватор, чекає розвантаження, " +  "потім повертається й стає в чергу на завантаження зерном від комбайнів. ");  dev.setText("Автомієнко Я. Г., Гречко І. О., ПI-221");   volumeSlider.valueProperty().addListener((obs, oldVal, newVal) -> {  if (newVal.doubleValue() == 0)  sound.setStyle("-fx-background-image: url('settings/soundClose.png');");  else  sound.setStyle("-fx-background-image: url('settings/sound.png');");  *musicControl*.setVolume(newVal.doubleValue());  });   volumeSlider.setValue(*musicControl*.getVolume());    topInfo.setOnMousePressed(event -> {  deltaX = event.getSceneX();  deltaY = event.getSceneY();  });   topInfo.setOnMouseDragged(event -> {  *mainStage*.setX(event.getScreenX() - deltaX);  *mainStage*.setY(event.getScreenY() - deltaY);  });   sound.setOnMousePressed(event -> {  sound.setStyle("-fx-background-image: url('settings/soundClose.png');");  volumeSlider.setValue(0.0);  *musicControl*.setVolume(0.0);  });   cls.setOnAction(event -> *mainStage*.fireEvent(new WindowEvent(*mainStage*, WindowEvent.*WINDOW\_CLOSE\_REQUEST*)));  } } |

В лістингу 4.5 наведено код класу MainWindow.

Лістинг 4.5 – Код класу MainWindow.

|  |
| --- |
| package controller;  import javafx.animation.TranslateTransition; import javafx.application.Platform; import javafx.fxml.FXML; import javafx.scene.Group; import javafx.scene.control.Button; import javafx.scene.control.Label; import javafx.scene.control.ScrollPane; import javafx.scene.image.Image; import javafx.scene.image.ImageView; import javafx.scene.input.ScrollEvent; import javafx.scene.layout.Pane; import javafx.stage.Stage; import javafx.stage.Window; import javafx.stage.WindowEvent; import javafx.util.Duration; import util.Giff; import util.View; import util.Timer; import util.entity.Combiner; import util.entity.CommandCentral; import util.entity.ProcessingPoint; import util.entity.Truck;  import java.util.\*; import java.util.concurrent.ExecutorService; import java.util.concurrent.Executors;  public class MainWindow {   private final ExecutorService combinerExec = Executors.*newCachedThreadPool*(r -> {  Thread a = new Thread(r);  a.setName("Combiner");  return a;  });  private final ExecutorService truckExec = Executors.*newCachedThreadPool*(r -> {  Thread a = new Thread(r);  a.setName("Truck");  return a;  });  private final ExecutorService processingPointExec = Executors.*newCachedThreadPool*(r -> {  Thread a = new Thread(r);  a.setName("PP");  return a;  });   // Main stage  private Window root;   // Main pane  @FXML  private Pane mainPane;   @FXML  private Group rightPanelGroup; // right panel and right panel gradient  @FXML  private ImageView rightPanel;  @FXML  private ImageView rightPanelGradient;   //right panel images  private final Image rightPanelImage = new Image(  Objects.*requireNonNull*(getClass().getResource("/icon/rightPanel.png")).toString()  );  private final Image rightPanelGradientImage = new Image(  Objects.*requireNonNull*(getClass().getResource("/icon/rightPanel.png")).toString()  //"file: x"  );   // Left panel  @FXML  private ImageView leftPanel;   // Left panel image  private final Image leftPanelImage = new Image(  Objects.*requireNonNull*(getClass().getResource("/icon/leftPanel.png")).toString()  );    // Icons  @FXML  private ImageView storageIcon;  @FXML  private ImageView truckIcon;  @FXML  private ImageView zernoIcon;   // icons images  private final Image storageIconImage = new Image(  Objects.*requireNonNull*(getClass().getResource("/icon/storageIcon.png")).toString()  );  private final Image truckIconImage = new Image(  Objects.*requireNonNull*(getClass().getResource("/icon/truckIcon.png")).toString()  );  private final Image zernoIconImage = new Image(  Objects.*requireNonNull*(getClass().getResource("/icon/zernoIcon.png")).toString()  );    // Amount  @FXML  private Label storageCount;  @FXML  private Label truckCount;  @FXML  private Label zerno;   @FXML  private Button cls;    // Scroll pane for timers  @FXML  private ScrollPane scrollPane;  @FXML  private Pane paneOnScroll;   // List for working timers  private final List<Timer> timerQueue = new ArrayList<>();   // Not modified list of combiners giff positions  public final List<View> views = new ArrayList<>();   // Animation for moving scroll pane  private TranslateTransition animation;  private boolean isOut;   // Command central  private CommandCentral center;   // Properties  private Properties properties;   public static Stage *mainStage*;  @FXML  private Pane topMain;  private double deltaX, deltaY;    // When clicked on the right panel  @FXML  private void animation() {  if (isOut)  animation.setToX(0);  else  animation.setToX(204);   animation.play();  }   // When scrolling  @FXML  private void onScroll(ScrollEvent event) {  double deltaY = event.getDeltaY();  double width = scrollPane.getContent().getBoundsInLocal().getWidth();  double vValue = scrollPane.getVvalue();  scrollPane.setVvalue(vValue - deltaY / width);  }    public void initialization() {  CommandCentral.*working\_mode* = true;  center = new CommandCentral(  this,  (int) properties.get("combinerCount"),  (int) properties.get("truckCount"),  (int) properties.get("PPCount")  );   root.setOnCloseRequest(event -> clean());   leftPanel.setImage(leftPanelImage);   rightPanel.setImage(rightPanelImage);  rightPanelGradient.setImage(rightPanelGradientImage);  rightPanelGradient.setViewOrder(0);   storageIcon.setImage(storageIconImage);  truckIcon.setImage(truckIconImage);  zernoIcon.setImage(zernoIconImage);   animation = new TranslateTransition(  Duration.*seconds*(1), rightPanelGroup  );  animation.setOnFinished(event -> isOut = !isOut);   topMain.setOnMousePressed(event -> {  deltaX = event.getSceneX();  deltaY = event.getSceneY();  });   topMain.setOnMouseDragged(event -> {  *mainStage*.setX(event.getScreenX() - deltaX);  *mainStage*.setY(event.getScreenY() - deltaY);  });   cls.setOnAction(event -> *mainStage*.fireEvent(new WindowEvent(*mainStage*, WindowEvent.*WINDOW\_CLOSE\_REQUEST*)));   initialize();  }   /////////////// Initialize ///////////////   private void initialize() {  initializeTrucks();  initializeProcessingPoints();  initializeCombiners();  }   // For stopping all timers  public final List<Truck> trucks = new ArrayList<>();   private void initializeTrucks() {  int count = (int) properties.get("truckCount");  for (int i = 0; i < count; i++) {  Truck truck = new Truck(  center,  (double) properties.get("toPPTime"),  (double) properties.get("toCombinersTime")  );  trucks.add(truck);  truckExec.execute(truck);  }  }   private void initializeProcessingPoints() {  int count = (int) properties.get("PPCount");  for (int i = 0; i < count; i++) {  ProcessingPoint PP = new ProcessingPoint(center, (double) properties.get("receptionTime"));  processingPointExec.execute(PP);  }  }   private void initializeCombiners() {  int count = (int) properties.get("combinerCount");  for (int i = 1; i <= count; i++) {  Combiner combiner = new Combiner(center, (double) properties.get("harvestTime"));  ImageView forCombiner = new ImageView();  ImageView forTruck = new ImageView();  int[] cords = setParePos(i);  forCombiner.setLayoutX(cords[0]);  forCombiner.setLayoutY(cords[1]);  forTruck.setLayoutX(cords[0]);  forTruck.setLayoutY(cords[1]);   mainPane.getChildren().addAll(forCombiner, forTruck);   View view = new View(combiner, forCombiner, forTruck);  views.add(view);   combinerExec.execute(combiner);  }  }   private int[] setParePos(int count) {  int[] cords = new int[2];  switch (count) {  case 1 -> {  return cords;  }  case 2 -> {  cords[0] = 228;  cords[1] = -111;  return cords;  }  case 3 -> {  cords[0] = -247;  cords[1] = 109;  return cords;  }  case 4 -> {  cords[0] = -490;  cords[1] = 223;  return cords;  }  case 5 -> {  cords[0] = 471;  cords[1] = -235;  return cords;  }  default -> throw new IllegalArgumentException(  "amount of combiners cannot be less than 1 or greater than 5"  );  }  }    /////////////// Properties and main stage ///////////////   public void setProperties(Properties properties) {  this.properties = properties;  }   public void setRoot(Window root) {  this.root = root;  }    /////////////// Timer ///////////////   public synchronized void remove(Timer timer) {  timerQueue.remove(timer);  Platform.*runLater*(() -> paneOnScroll.getChildren().remove(timer.getPane()));  regroup();  }   public synchronized void add(Timer timer) {  Platform.*runLater*(() -> {  timerQueue.add(timer);  timer.setY(timerQueue.indexOf(timer) \* 84 + 20);  paneOnScroll.getChildren().add(timer.getPane());  });  }   private void regroup() {  for (int i = 0; i < timerQueue.size(); i++) {  int finalI = i;  Platform.*runLater*(() -> timerQueue.get(finalI).setY(finalI \* 84 + 20));  }  }    /////////////// Set info ///////////////   public void setStorageCount(int count) {  String text = count + "/" + properties.get("PPCount");  Platform.*runLater*(() -> storageCount.setText(text));  }   public void setTruckCount(int count) {  String text = count + "/" + properties.get("truckCount");  Platform.runLater(() -> truckCount.setText(text));  }   public void setZerno(int count) {  String text = count + "т.";  Platform.runLater(() -> zerno.setText(text));  }    /////////////// Clean ///////////////   private void clean() {  CommandCentral.working\_mode = false;   Giff.stop();   truckExec.shutdownNow();  processingPointExec.shutdownNow();  combinerExec.shutdownNow();   Combiner.reset();  Truck.reset();  ProcessingPoint.reset();  }  } |

В лістингу 4.6 наведено код класу Giff.

Лістинг 4.6 – Код класу Giff.

|  |
| --- |
| package util;  import javafx.application.Platform; import javafx.scene.image.Image; import javafx.scene.image.ImageView;  import java.time.Duration; import java.util.ArrayList; import java.util.List; import java.util.concurrent.TimeUnit;  public class Giff {   private final static List<Giff> *giffs* = new ArrayList<>();   private Image[] images;   private ImageView view;   private volatile boolean stop;   private final Thread playing;   private boolean isEnd;   private boolean isCycle;   private long timeBetween;    {  playing = new Thread(() -> {  do {  for (int i = 0; i < images.length; i++) {  while (stop)  Thread.*onSpinWait*();  try {  TimeUnit.MICROSECONDS.sleep(timeBetween);  } catch (InterruptedException e) {  throw new RuntimeException(e);  }  int finalI = i;  Platform.*runLater*(() -> view.setImage(images[finalI]));  }  } while (isCycle);  isEnd = true;  });  playing.setName("Giff");  }    public Giff(Image[] frames) {  setFrames(frames);  *giffs*.add(this);  }    public static void stop() {  *giffs*.forEach(giff -> {  giff.stop = false;  giff.isCycle = false;  });  *giffs*.clear();  }    public void setTime(double timeInSeconds) {  timeBetween = (long) (timeInSeconds \* 1\_000\_000 / images.length);  }   public void setCycleMode(boolean isCycle) {  this.isCycle = isCycle;  }   public void setView(ImageView view) {  this.view = view;  }   public void setFrames(Image[] frames) {  images = frames;  }    public void play() {  stop = false;  playing.start();  }    public boolean isEnd() {  return isEnd;  }  } |

В лістингу 4.7 наведено код класу View.

Лістинг 4.7 – Код класу View.

|  |
| --- |
| package util;  import controller.Loading; import javafx.scene.image.ImageView; import util.entity.Combiner; import util.entity.Truck;  public class View {   private final ImageView viewForTruck;   private final Giff combinerGiff;    public View(Combiner combiner, ImageView forCombiner, ImageView forTruck) {  viewForTruck = forTruck;   combinerGiff = new Giff(Loading.*COMBINER\_FRAMES*);   //start combiner giff  combinerGiff.setTime(1);  combinerGiff.setView(forCombiner);  combinerGiff.setCycleMode(true);  combinerGiff.play();   combiner.setWhenBegunFull(() -> combinerGiff.setFrames(Loading.*COMBINER\_LIGHT\_BULB\_FRAMES*));  combiner.setWhenBegunEmpty(() -> combinerGiff.setFrames(Loading.*COMBINER\_FRAMES*));  }    public void playTruck(Truck truck) {  Giff truckGiff = new Giff(Loading.*TRUCK\_FRAMES*);  truckGiff.setTime(5);  truckGiff.setView(viewForTruck);  truck.setGiff(truckGiff);  truckGiff.play();  }  } |

В лістингу 4.8 наведено код класу Timer.

Лістинг 4.8 – Код класу Timer.

|  |
| --- |
| package util;  import javafx.animation.Transition; import javafx.animation.TranslateTransition; import javafx.application.Platform; import javafx.beans.property.BooleanProperty; import javafx.beans.property.SimpleBooleanProperty; import javafx.scene.control.Label; import javafx.scene.layout.Pane; import javafx.scene.media.Media; import javafx.scene.media.MediaPlayer; import javafx.scene.media.MediaView; import javafx.scene.text.Font; import javafx.util.Duration; import util.entity.CommandCentral;  import java.util.Objects;  public class Timer {   private final Pane pane;    private static final Media *start* = new Media(  Objects.*requireNonNull*(Timer.class.getResource("/timer/timerStart.mp4")).toString()  );   private static final Media *playRed* = new Media(  Objects.*requireNonNull*(Timer.class.getResource("/timer/timerRed.mp4")).toString()  );  private static final Media *playGreen* = new Media(  Objects.*requireNonNull*(Timer.class.getResource("/timer/timerGreen.mp4")).toString()  );  private static final Media *playBlue* = new Media(  Objects.*requireNonNull*(Timer.class.getResource("/timer/timerBlue.mp4")).toString()  );    private static final Media *endRed* = new Media(  Objects.*requireNonNull*(Timer.class.getResource("/timer/timerStopR.mp4")).toString()  );  private static final Media *endGreen* = new Media(  Objects.*requireNonNull*(Timer.class.getResource("/timer/timerStopG.mp4")).toString()  );  private static final Media *endBlue* = new Media(  Objects.*requireNonNull*(Timer.class.getResource("/timer/timerStopB.mp4")).toString()  );    private final MediaPlayer startMP = new MediaPlayer(*start*);  private final MediaPlayer playRedMP = new MediaPlayer(*playRed*);  private final MediaPlayer playGreenMP = new MediaPlayer(*playGreen*);  private final MediaPlayer playBlueMP = new MediaPlayer(*playBlue*);  private final MediaPlayer endRedMP = new MediaPlayer(*endRed*);  private final MediaPlayer endGreenMP = new MediaPlayer(*endGreen*);  private final MediaPlayer endBlueMP = new MediaPlayer(*endBlue*);   private final MediaView view = new MediaView();     private final Label label = new Label();   private final TranslateTransition inStart;  private final TranslateTransition inEnd;    private Mode[] modes;   private double time;   public Timer() {   pane = new Pane();  pane.setLayoutX(210);  pane.setPrefSize(208, 64);  pane.setStyle("-fx-background-color: rgba(0, 0, 0, 0)");  pane.getChildren().addAll(view, label);   inStart = new TranslateTransition(Duration.*seconds*(0.5), pane);  inStart.setToX(-208);   inEnd = new TranslateTransition(Duration.*seconds*(0.5), pane);  inEnd.setToX(208);   label.setLayoutY(17.5);  label.setLayoutX(75);  label.setViewOrder(-1);  label.setFont(new Font("Calibri Bold", 12.5));  label.setTextFill(javafx.scene.paint.Color.*valueOf*("#FFF0C6"));   }   private void play(Mode mode) {  double maxTime = 24;  switch (mode) {  case *START* -> {  startMP.seek(Duration.*ZERO*);  view.setMediaPlayer(startMP);  playAnimationQuery(inStart);  playMediaQuery(startMP);  }  case *PLAYING* -> {  MediaPlayer play = getPlaying();  play.seek(Duration.*ZERO*);  view.setMediaPlayer(play);  play.setRate(maxTime / time);  playMediaQuery(play);  }  case *END* -> {  MediaPlayer end = getEnding();  end.seek(Duration.*ZERO*);  view.setMediaPlayer(end);  playMediaQuery(end);  playAnimationQuery(inEnd);  }  }  }   public void setDescription(String description) {  Platform.*runLater*(() -> label.setText(description));  }    public void play() {  for (Mode mode : modes) {  if (!CommandCentral.*working\_mode*)  return;  play(mode);  }  }   private Color color;   public void setColor(Color color) {  this.color = color;  }   private MediaPlayer getPlaying() {  return switch (color) {  case *RED* -> playRedMP;  case *GREEN* -> playGreenMP;  case *BLUE* -> playBlueMP;  };  }   private MediaPlayer getEnding() {  return switch (color) {  case *RED* -> endRedMP;  case *GREEN* -> endGreenMP;  case *BLUE* -> endBlueMP;  };  }   public void setModes(Mode... modes) {  this.modes = modes;  }   public void setTime(double time) {  this.time = time;  }   public void setY(double y) {  pane.setLayoutY(y);  }   public Pane getPane() {  return pane;  }   private void playAnimationQuery(Transition transition) {  if (!CommandCentral.*working\_mode*)  return;   BooleanProperty isFinished = new SimpleBooleanProperty(false);  transition.setOnFinished(e -> isFinished.setValue(true));  Platform.*runLater*(transition::play);   while (!isFinished.get()) {  if (!CommandCentral.*working\_mode*) {  transition.stop();  return;  }  Thread.*onSpinWait*();  }   transition.setOnFinished(null);   }   private void playMediaQuery(MediaPlayer mp) {  BooleanProperty isFinished = new SimpleBooleanProperty(false);  mp.setOnEndOfMedia(() -> isFinished.setValue(true));  Platform.*runLater*(mp::play);   while(!isFinished.get()) {  if (!CommandCentral.*working\_mode*)  return;  Thread.*onSpinWait*();  }   }   public void stop() {   inStart.stop();  inEnd.stop();   startMP.stop();  playRedMP.stop();  playGreenMP.stop();  playBlueMP.stop();  endRedMP.stop();  endGreenMP.stop();  endBlueMP.stop();   startMP.dispose();  playRedMP.dispose();  playGreenMP.dispose();  playBlueMP.dispose();  endRedMP.dispose();  endGreenMP.dispose();  endBlueMP.dispose();   }   public enum Mode {  *START*,  *PLAYING*,  *END* }   public enum Color {  *RED*,  *GREEN*,  *BLUE* }  } |

В лістингу 4.9 наведено код класу CommandCentral.

Лістинг 4.9 – Код класу CommandCentral.

|  |
| --- |
| package util.entity;  import controller.MainWindow; import util.Timer;  import java.io.Serializable; import java.util.concurrent.ArrayBlockingQueue; import java.util.concurrent.BlockingQueue; import java.util.concurrent.atomic.AtomicInteger;  public class CommandCentral implements Serializable {   private static final long *serialVersionUID* = 1248379652L;   private final Object combinerMonitor = new Object();  private final Object machineMonitor = new Object();  private final Object PPMonitor = new Object();   public final MainWindow controller;   private final BlockingQueue<Combiner> fullCombinerBlockingQueue;   private final BlockingQueue<Truck> emptyTruckBlockingQueue;   private final BlockingQueue<Truck> fullTruckBlockingQueue;   private final BlockingQueue<ProcessingPoint> freePPBlockingQueue;   public CommandCentral(MainWindow controller, int combinerCount, int truckCount, int PPCount) {  this.controller = controller;  fullCombinerBlockingQueue = new ArrayBlockingQueue<>(combinerCount);  emptyTruckBlockingQueue = new ArrayBlockingQueue<>(truckCount);  fullTruckBlockingQueue = new ArrayBlockingQueue<>(truckCount);  freePPBlockingQueue = new ArrayBlockingQueue<>(PPCount);  }   private final AtomicInteger count = new AtomicInteger(0);     public static boolean *working\_mode* = true;     public void add(Combiner fullCombiner) {  synchronized (combinerMonitor) {  if (!emptyTruckBlockingQueue.isEmpty()) {  Truck emptyTruck = emptyTruckBlockingQueue.remove();  controller.setTruckCount(emptyTruckBlockingQueue.size());  emptyTruck.setCombiner(fullCombiner);  controller.views.get(fullCombiner.ID).playTruck(emptyTruck);  synchronized (emptyTruck.monitor) {  emptyTruck.monitor.notify();  }  } else  fullCombinerBlockingQueue.add(fullCombiner);   }  }   public void add(Truck truck, boolean isFull) {  synchronized (machineMonitor) {  if (isFull) {  if (!freePPBlockingQueue.isEmpty()) {  ProcessingPoint PP = freePPBlockingQueue.remove();  controller.setStorageCount(freePPBlockingQueue.size());  PP.setTruck(truck);  truck.TIMER.setColor(Timer.Color.*BLUE*);  truck.TIMER.setModes(Timer.Mode.*PLAYING*);  truck.TIMER.setDescription("#" + truck.ID + " Розвантаження...");  synchronized (PP.monitor) {  PP.monitor.notify();  }  } else {  truck.TIMER.setModes(Timer.Mode.*END*);  truck.TIMER.play();  controller.remove(truck.TIMER);  fullTruckBlockingQueue.add(truck);  }   } else {  if (!fullCombinerBlockingQueue.isEmpty()) {  Combiner combiner = fullCombinerBlockingQueue.remove();  truck.setCombiner(combiner);  controller.views.get(combiner.ID).playTruck(truck);  } else {  emptyTruckBlockingQueue.add(truck);  controller.setTruckCount(emptyTruckBlockingQueue.size());  }  }  }  }   public void add(ProcessingPoint freePP) {  synchronized (PPMonitor) {  if (!fullTruckBlockingQueue.isEmpty()) {  Truck fullTruck = fullTruckBlockingQueue.remove();  freePP.setTruck(fullTruck);  fullTruck.TIMER.setColor(Timer.Color.*BLUE*);  fullTruck.TIMER.setModes(Timer.Mode.*START*, Timer.Mode.*PLAYING*);  fullTruck.TIMER.setDescription("#" + fullTruck.ID + " Розвантаження...");  controller.add(fullTruck.TIMER);  synchronized (freePP.monitor) {  freePP.monitor.notify();  }  } else {  freePPBlockingQueue.add(freePP);  controller.setStorageCount(freePPBlockingQueue.size());  }  }  }   public void increment() {  count.incrementAndGet();  controller.setZerno(count.get());  }   } |

В лістингу 4.10 наведено код класу Combiner.

Лістинг 4.10 – Код класу Combiner.

|  |
| --- |
| package util.entity;  import java.time.Duration; import java.util.Objects;  public class Combiner extends Monitor implements Runnable {   public static final Combiner *NULL\_COMBINER* = new Combiner(null, 0);   private static int *count* = 0;  public final int ID;   private final double harvestTimeSeconds;   private final CommandCentral center;   private Runnable whenBegunFull;  private Runnable whenBegunEmpty;   public boolean isFull;    public Combiner(CommandCentral center, double harvestTimeSeconds) {  if (center == null)  ID = -1;  else  ID = *count*++;  this.center = center;  this.harvestTimeSeconds = harvestTimeSeconds;  }    public void setWhenBegunFull(Runnable value) {  whenBegunFull = value;  }   public void setWhenBegunEmpty(Runnable value) {  whenBegunEmpty = value;  }   public static void reset() {  *count* = 0;  }    @Override  public void run() {  while (CommandCentral.*working\_mode*)  synchronized (monitor) {  try {  while (isFull)  monitor.wait();   whenBegunEmpty.run();   double millis = harvestTimeSeconds \* 1000;  Thread.*sleep*(Duration.*ofMillis*((int) millis));   center.add(this);  isFull = true;   whenBegunFull.run();  } catch (InterruptedException e) {  Thread.*currentThread*().interrupt();  }  }  }   @Override  public boolean equals(Object o) {  if (this == o)  return true;  if (o == null || getClass() != o.getClass())  return false;   Combiner combiner = (Combiner) o;  return ID == combiner.ID;  }   @Override  public int hashCode() {  return Objects.*hash*(ID);  }   @Override  public String toString() {  return "Combiner #" + ID;  }  } |

В лістингу 4.11 наведено код класу Truck.

Лістинг 4.11 – Код класу Truck.

|  |
| --- |
| package util.entity;  import util.Giff; import util.Timer;  import java.time.Duration;  public class Truck extends Monitor implements Runnable {   private static int *count*;  public final int ID = ++*count*;   private final double toCombinersTimeSeconds;  private final double toPPTimeSeconds;   private final CommandCentral center;   public final Timer TIMER = new Timer();   public boolean isFull;   private Giff giff;   private Combiner combiner;    public Truck(CommandCentral center, double toPPTimeSeconds,  double toCombinersTimeSeconds) {  this.center = center;  this.center.add(this, false);  this.toPPTimeSeconds = toPPTimeSeconds;  this.toCombinersTimeSeconds = toCombinersTimeSeconds;  }    public void setGiff(Giff giff) {  this.giff = giff;  }   public void setCombiner(Combiner combiner) {  this.combiner = combiner;  }   public static void reset() {  *count* = 0;  }    private void startUnloadCombine() {  try {  //unloading combiner time to synchronize with giff  Thread.*sleep*(Duration.*ofSeconds*(4));  combiner.isFull = false;  isFull = true;  synchronized (combiner.monitor) {  combiner.monitor.notify();  }  while (!giff.isEnd())  Thread.*onSpinWait*();  giff = null;  } catch (InterruptedException e) {  TIMER.stop();  Thread.*currentThread*().interrupt();  }  }    @Override  public void run() {  while (CommandCentral.*working\_mode*) {   synchronized (monitor) {  try {  while (combiner == null)  monitor.wait();   if (!CommandCentral.*working\_mode*)  break;   if (combiner.equals(Combiner.*NULL\_COMBINER*)) {  combiner = null;  TIMER.setTime(toCombinersTimeSeconds);  TIMER.setColor(Timer.Color.*GREEN*);  TIMER.setModes(Timer.Mode.*PLAYING*, Timer.Mode.*END*);  TIMER.setDescription("#" + ID + " їде до поля");  TIMER.play();  center.controller.remove(TIMER);  center.add(this, false);  } else if (!isFull) {  startUnloadCombine();   combiner = null;   TIMER.setTime(toPPTimeSeconds);  TIMER.setColor(Timer.Color.*RED*);  TIMER.setModes(Timer.Mode.*START*, Timer.Mode.*PLAYING*);  TIMER.setDescription("#" + ID + " їде до складу");  center.controller.add(TIMER);  TIMER.play();   center.add(this, true);  }  } catch (InterruptedException e) {  TIMER.stop();  Thread.*currentThread*().interrupt();  return;  }  }  }  TIMER.stop();  }   @Override  public String toString() {  return "Truck #" + ID;  } } |

В лістингу 4.12 наведено код класу ProcessingPoint.

Лістинг 4.12 – Код класу ProcessingPoint.

|  |
| --- |
| package util.entity;  public class ProcessingPoint extends Monitor implements Runnable {   private static int *count*;  public final int ID = ++*count*;   private final double receptionTimeSeconds;   private final CommandCentral center;   private Truck processedTruck;   public ProcessingPoint(CommandCentral center, double receptionTimeSeconds) {  this.center = center;  this.center.add(this);  this.receptionTimeSeconds = receptionTimeSeconds;  }    public void setTruck(Truck fullTruck) {  processedTruck = fullTruck;  }   public static void reset() {  *count* = 0;  }    private void unloadingTruck() {  processedTruck.isFull = false;  processedTruck.setCombiner(Combiner.*NULL\_COMBINER*);  synchronized (processedTruck.monitor) {  processedTruck.monitor.notify();  }  center.increment();  }    @Override  public void run() {  while (CommandCentral.*working\_mode*) {   synchronized (monitor) {  try {  while (processedTruck == null) {  monitor.wait();  }  //unloading full truck  processedTruck.TIMER.setTime(receptionTimeSeconds);  processedTruck.TIMER.play(); //wait  unloadingTruck();  //PP is now free  processedTruck = null;  center.add(this);  } catch (InterruptedException e) {  Thread.*currentThread*().interrupt();  }  }   }  }   @Override  public String toString() {  return "ProcessingPoint #" + ID;  } } |

В лістингу 4.13 наведено код класу Monitor.

Лістинг 4.13 – Код класу Monitor.

|  |
| --- |
| package util.entity;  public class Monitor {  public final Object monitor = new Object(); } |

В лістингу 4.14 наведено код класу Runner.

Лістинг 4.14 – Код класу Runner.

|  |
| --- |
| public class Runner {  public static void main(String[] args) {  Launcher.*main*(args);  } } |

# ТЕСТУВАННЯ ПРОЄКТУ

На рисунку 4.1 зображено вікно завантаження програми.



Рисунок 4.1 – Вікно завантаження програми

На рисунку 4.2 зображено вікно налаштувань.



Рисунок 4.2 – Вікно налаштувань

На рисунку 4.3 зображено вікно інформації про програму.

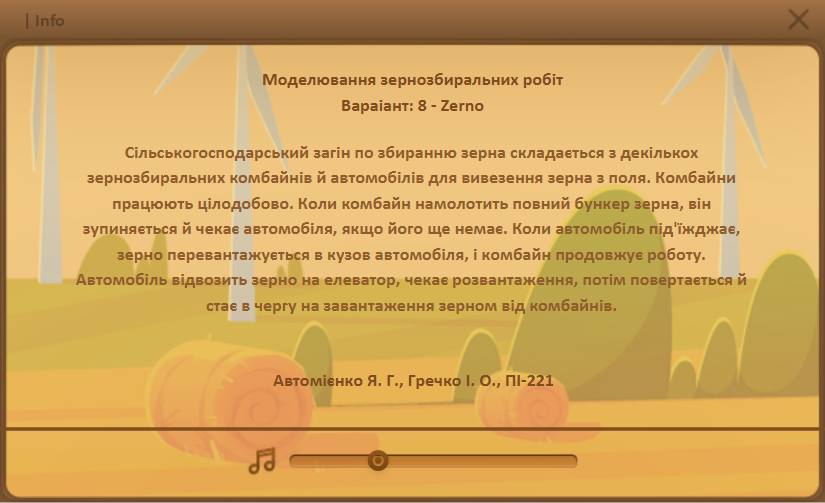


Рисунок 4.3 – Вікно інформації про програму

На рисунках 4.4-4.5 зображено роботу програми.

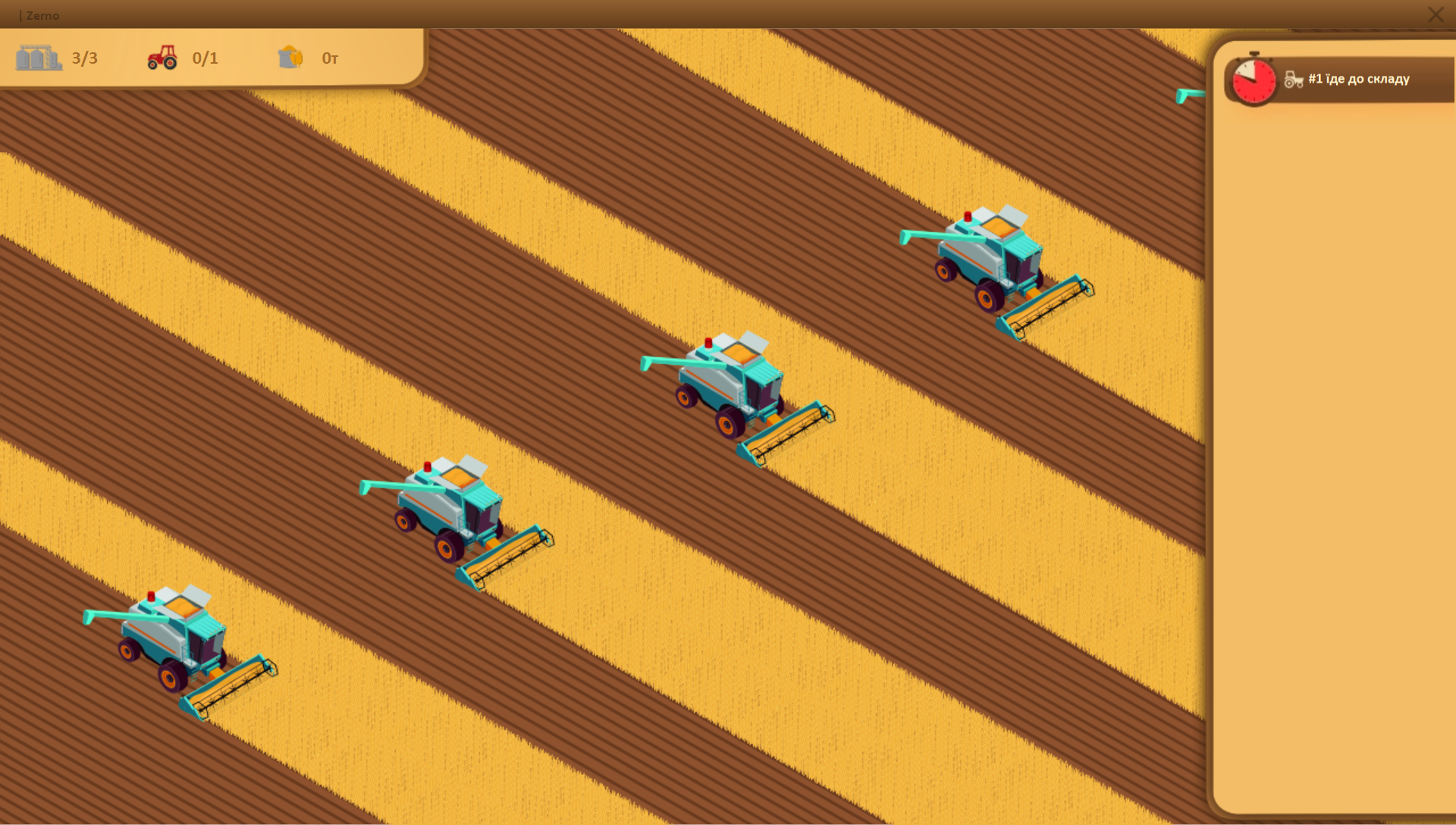


Рисунок 4.4 – Вікно роботи програми (з панеллю)



Рисунок 4.5 – Вікно роботи програми (без панелі)

# ВИСНОВКИ

У результаті виконання курсового проєкту було розроблено інформаційно-довідкову систему «Zerno». Система створена у вигляді організації об’єктів, що працюють паралельно. Розроблений проєкт повністю реалізує поставлені перед ним завдання. Було створено класи моделей, такі як Combiner, Truck та ProcessingPoint, а також реалізований додатковий клас для логіки CommandCentral. В проєкті був реалізований інтерфейс, який є простим і зрозумілим у використанні. Він підібраний відповідно до сучасних тенденцій.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Simon Kendal. Object oriented programming using Java. Ventus Publishing ApS-, 2009. – 209 c.

2. Програмування мовою Java. Олексій Васильєв. Видавництво: Навчальна книга - Богдан, 2020. -696 c.

3. Копитко М.Ф., Іванків К.С. Основи програмування мовою Java: Тексти лекцій. − Львів: Видавничий центр ЛНУ ім. Івана Франка, 2002.− 83 с.

4. Shildt Herbert. Java Complete guide. Eleventh edition. — McGraw-Hill Education, 2018. — 1248 р.

5. Object-Oriented Analysis and Design with Applications (3rd Edition) by Grady Booch. Addison-Wesley Professional, 2007

Допоміжна

6. Bloch Joshua. Effective Java. Third Edition. — Addison-Wesley, 2018. — 413 p.

7. Goetz Brain. Java concurrency in practice. Addison Wesley, 2010.