МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №6**

**Дисциплина: Платформо-независимое программирование**

Работу выполнил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Воробьев А.О.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т. А. Приходько

Краснодар

2025

**Тема**: Потоки в Java.

**Задание:**

**Лото.** Четыре игрока строго по очереди выставляют по одной свей фишке (у каждого свой цвет фишки) на общую доску, раскрашенную в разные цвета, о чем каждый выдает сообщение, например: «Красный, фишка 3». По окончании (когда все клетки заполнены) сообщается, кто закончил первым. Фишек по 10 у каждого, мест на доске 30. Расположение цветов на доске случайное. (Графический вариант).

**Лото1.** Предыдущая задача, но порядок игроков и размещение фишек случайное, у каждого из 4-х игроков фишки одного цвета. Есть графическое поле, на которое все игроки должны попытаться выставить свои 10 фишек в случайные места. Но мест всего 30. Побеждает тот, кто выставит больше

**Ход работы:**

В ходе работы создаются следующие классы:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

В классе GameSettings задаются основные настройки игры, например, количество игроков, количество фишек у каждого игрока и т.д. Листинг:

import java.awt.\*;  
  
public class GameSettings {  
 public static final int *PLAYERS\_COUNT* = 4;  
 public static final int *TOKENS\_PER\_PLAYER* = 14;  
 public static final int *BOARD\_ROWS* = 6;  
 public static final int *BOARD\_COLS* = 8;  
 public static final int *BOARD\_SIZE* = *BOARD\_ROWS* \* *BOARD\_COLS*;  
 public static final Color[] *PLAYER\_COLORS* = {  
 Color.*RED*, Color.*BLUE*, Color.*GREEN*, Color.*YELLOW* };  
  
 private GameMode mode;  
  
 public GameSettings(GameMode mode) {  
 this.mode = mode;  
 }  
  
 public GameMode getMode() {  
 return mode;  
 }  
  
 public void setMode(GameMode mode) {  
 this.mode = mode;  
 }  
}

Класс GameController используется для получения следующего игрока и перезапуска игры:

import java.util.concurrent.atomic.AtomicInteger;  
  
public class GameController {  
 private static final AtomicInteger *currentPlayer* = new AtomicInteger(0);  
  
 public static int getCurrentPlayer() {  
 return *currentPlayer*.get();  
 }  
  
 public static void nextPlayer() {  
 *currentPlayer*.set((*currentPlayer*.get() + 1) % GameSettings.*PLAYERS\_COUNT*);  
 }  
  
 public static void resetGame() {  
 *currentPlayer*.set(0);  
 }  
}

Класс GameBoard создает основное поле клеток и определяет некоторую логику:

import javax.swing.\*;  
import java.awt.\*;  
import java.util.Random;  
  
public class GameBoard extends JPanel {  
 public static final int *BOARD\_SIZE* = 30;  
 public static final Color[] *BOARD\_COLORS* = GameSettings.*PLAYER\_COLORS*;  
  
 private final JLabel[] cells = new JLabel[*BOARD\_SIZE*];  
  
 private int[] playerScores = new int[GameSettings.*PLAYERS\_COUNT*];  
  
 public GameBoard() {  
 setLayout(new GridLayout(5, 6));  
 setBorder(BorderFactory.*createEmptyBorder*(10, 10, 10, 10)); // Отступы  
 initializeCells();  
 }  
  
 private void initializeCells() {  
 Random random = new Random();  
 for (int i = 0; i < *BOARD\_SIZE*; i++) {  
 cells[i] = new JLabel("", SwingConstants.*CENTER*);  
 cells[i].setOpaque(true);  
 cells[i].setBackground(*BOARD\_COLORS*[random.nextInt(*BOARD\_COLORS*.length)]);  
 cells[i].setBorder(BorderFactory.*createLineBorder*(Color.*BLACK*));  
 add(cells[i]);  
 }  
 }  
  
 public synchronized boolean placeToken(int cellIndex, int playerId, Color playerColor, GameMode mode) {  
 if (cells[cellIndex].getText().isEmpty()) {  
 // В последовательном режиме проверяем цвет клетки  
 if (mode == GameMode.*SEQUENTIAL* && !cells[cellIndex].getBackground().equals(playerColor)) {  
 return false;  
 }  
  
 cells[cellIndex].setText(String.*valueOf*(playerId + 1));  
 cells[cellIndex].setForeground(Color.*BLACK*); // Фишки всегда черные для лучшей видимости  
 playerScores[playerId]++;  
 return true;  
 }  
 return false;  
 }  
  
 public boolean isBoardFull() {  
 for (JLabel cell : cells) {  
 if (cell.getText().isEmpty()) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 public synchronized void resetBoard() {  
 for (int i = 0; i < *BOARD\_SIZE*; i++) {  
 cells[i].setText("");  
 playerScores = new int[GameSettings.*PLAYERS\_COUNT*];  
 }  
 Random random = new Random();  
 for (JLabel cell : cells) {  
 cell.setBackground(*BOARD\_COLORS*[random.nextInt(*BOARD\_COLORS*.length)]);  
 }  
 }  
  
 public synchronized boolean isCellEmpty(int cellIndex) {  
 return cells[cellIndex].getText().isEmpty();  
 }  
  
 public synchronized Color getCellColor(int cellIndex) {  
 return cells[cellIndex].getBackground();  
 }  
  
 public int[] getPlayerScores() {  
 return playerScores.clone();  
 }  
  
 public void announceWinner(GameLog log) {  
 int maxScore = -1;  
 int winnerId = -1;  
  
 for (int i = 0; i < playerScores.length; i++) {  
 if (playerScores[i] > maxScore) {  
 maxScore = playerScores[i];  
 winnerId = i;  
 }  
 }  
  
 if (winnerId != -1) {  
 log.addMessage("Победитель: Игрок " + (winnerId + 1) + " (" + getColorName(GameSettings.*PLAYER\_COLORS*[winnerId]) +  
 ") с " + maxScore + " очками!");  
 }  
 }  
  
 private String getColorName(Color color) {  
 if (color.equals(Color.*RED*)) return "Красный";  
 if (color.equals(Color.*BLUE*)) return "Синий";  
 if (color.equals(Color.*GREEN*)) return "Зеленый";  
 if (color.equals(Color.*YELLOW*)) return "Желтый";  
 return "Неизвестный";  
 }  
}

Класс GameLog отвечает за вывод результатов и хода игры.

Класс GameFrame непосредственно запускает игру и создает игроков (потоки):

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
  
public class GameFrame extends JFrame {  
 private final GameBoard board;  
 private final GameLog log;  
 private final GameSettings settings;  
 private static Player[] *players*;  
  
 public GameFrame() {  
 settings = new GameSettings(GameMode.*SEQUENTIAL*);  
 setTitle("Лото");  
 setSize(900, 600);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLayout(new BorderLayout());  
  
 board = new GameBoard();  
 log = new GameLog();  
  
 // Инициализация игроков с передачей settings  
 *players* = new Player[GameSettings.*PLAYERS\_COUNT*];  
 for (int i = 0; i < *players*.length; i++) {  
 *players*[i] = new Player(i, board, log, settings);  
 }  
  
 // Панель управления  
 JPanel controlPanel = new JPanel();  
 JButton switchModeBtn = new JButton("Переключить режим");  
 switchModeBtn.addActionListener(this::switchGameMode);  
 controlPanel.add(switchModeBtn);  
  
 JButton startBtn = new JButton("Начать игру");  
 startBtn.addActionListener(e -> startGame());  
 controlPanel.add(startBtn);  
  
 // Основная панель  
 JPanel mainPanel = new JPanel(new BorderLayout());  
 mainPanel.add(board, BorderLayout.*CENTER*);  
 mainPanel.add(log, BorderLayout.*EAST*);  
 mainPanel.add(controlPanel, BorderLayout.*SOUTH*);  
  
 add(mainPanel);  
 }  
  
 private void startGame() {  
 stopCurrentPlayers();  
 GameController.*resetGame*();  
 log.clear();  
 board.resetBoard();  
 initializePlayers();  
  
 for (Player player : *players*) {  
 player.start();  
 }  
 }  
  
 public static Player[] getPlayers() {  
 return *players*;  
 }  
  
 private void stopCurrentPlayers() {  
 if (*players* != null) {  
 for (Player player : *players*) {  
 if (player != null) {  
 player.stopPlayer();  
 }  
 }  
 }  
 }  
  
 private void initializePlayers() {  
 *players* = new Player[GameSettings.*PLAYERS\_COUNT*];  
 for (int i = 0; i < *players*.length; i++) {  
 *players*[i] = new Player(i, board, log, settings);  
 }  
 }  
  
 private void switchGameMode(ActionEvent e) {  
 GameMode newMode = settings.getMode() == GameMode.*SEQUENTIAL* ? GameMode.*RANDOM* : GameMode.*SEQUENTIAL*;  
 settings.setMode(newMode);  
 log.clear();  
 log.addMessage("Режим изменён на: " +  
 (newMode == GameMode.*SEQUENTIAL* ? "Последовательный" : "Случайный"));  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> {  
 GameFrame game = new GameFrame();  
 game.setVisible(true);  
 });  
 }  
}

Класс Player реализует игрока, который является потоком. Для этого он наследуется от класса Thread. Внутри него содержится volatile переменная running, то есть любой поток может иметь к ней доступ. Для реализации функционала переопределяется метод run(), в котором определяется основная логика потоков. Для приостановки игрока используется sleep(), а для полной остановки – interrupt().

import java.awt.Color;  
import java.util.Random;  
  
public class Player extends Thread {  
 private final int playerId;  
 private final GameBoard board;  
 private final GameLog log;  
 private final GameSettings settings;  
 private final Random random = new Random();  
 private volatile boolean running = true;  
 private int tokensPlaced = 0;  
 private int lastCellIndex = -1; // Для последовательного режима  
  
 public Player(int playerId, GameBoard board, GameLog log, GameSettings settings) {  
 this.playerId = playerId;  
 this.board = board;  
 this.log = log;  
 this.settings = settings;  
 }  
  
 public void stopPlayer() {  
 running = false;  
 this.interrupt();  
 }  
  
 @Override  
 public void run() {  
 while (running && tokensPlaced < GameSettings.*TOKENS\_PER\_PLAYER* && !board.isBoardFull()) {  
 if (settings.getMode() == GameMode.*SEQUENTIAL*) {  
 placeSequentialToken();  
 } else {  
 placeRandomToken();  
 }  
  
 try {  
 Thread.*sleep*(100);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 return;  
 }  
 }  
  
 if (tokensPlaced >= GameSettings.*TOKENS\_PER\_PLAYER* || board.isBoardFull()) {  
 checkGameEnd();  
 }  
 }  
  
 private boolean allPlayersFinished() {  
 // Простая проверка - в реальной игре нужно более сложное решение  
 for (Player player : GameFrame.*getPlayers*()) {  
 if (player.tokensPlaced < GameSettings.*TOKENS\_PER\_PLAYER*) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 private void placeSequentialToken() {  
 synchronized (board) {  
 try {  
 while (GameController.*getCurrentPlayer*() != playerId && !board.isBoardFull()) {  
 board.wait();  
 }  
 if (board.isBoardFull()) {  
 checkGameEnd();  
 return;  
 }  
  
 int cellIndex = findNextEmptyCellOfMyColor();  
  
 if (cellIndex == -1) {  
 // Не нашли подходящую клетку - пропускаем ход  
 log.addMessage(String.*format*("Игрок %d (%s) не нашел подходящую клетку - пропуск хода",  
 playerId + 1, getColorName()));  
  
 GameController.*nextPlayer*();  
 board.notifyAll();  
 return;  
 }  
  
 if (board.placeToken(cellIndex, playerId,  
 GameSettings.*PLAYER\_COLORS*[playerId], settings.getMode())) {  
  
 tokensPlaced++;  
 lastCellIndex = cellIndex;  
 log.addMessage(String.*format*("Игрок %d (%s), фишка %d в ячейку %d",  
 playerId + 1, getColorName(), tokensPlaced, cellIndex + 1));  
  
 if (tokensPlaced >= GameSettings.*TOKENS\_PER\_PLAYER*) {  
 checkGameEnd();  
 }  
  
 GameController.*nextPlayer*();  
 board.notifyAll();  
 }  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 }  
  
 private void placeRandomToken() {  
 int attempts = 0;  
 while (attempts < GameSettings.*BOARD\_SIZE* \* 2) {  
 int cellIndex = random.nextInt(GameSettings.*BOARD\_SIZE*);  
 if (board.placeToken(cellIndex, playerId, GameSettings.*PLAYER\_COLORS*[playerId], settings.getMode())) {  
 tokensPlaced++;  
 log.addMessage(String.*format*("Игрок %d (%s), фишка %d в ячейку %d",  
 playerId + 1, getColorName(), tokensPlaced, cellIndex + 1));  
 return;  
 }  
 attempts++;  
 }  
 }  
  
 private int findNextEmptyCellOfMyColor() {  
 Color myColor = GameSettings.*PLAYER\_COLORS*[playerId];  
 int attempts = 0;  
  
 while (attempts < GameBoard.*BOARD\_SIZE*) {  
 int currentIndex = (lastCellIndex + 1 + attempts) % GameBoard.*BOARD\_SIZE*;  
  
 if (board.isCellEmpty(currentIndex)) {  
 if (settings.getMode() == GameMode.*SEQUENTIAL*) {  
 if (board.getCellColor(currentIndex).equals(myColor)) {  
 return currentIndex;  
 }  
 } else {  
 return currentIndex;  
 }  
 }  
 attempts++;  
 }  
  
 return -1;  
 }  
  
 private void checkGameEnd() {  
 // Проверяем, все ли игроки завершили ходить  
 boolean allFinished = true;  
 for (Player player : GameFrame.*getPlayers*()) {  
 if (player.tokensPlaced < GameSettings.*TOKENS\_PER\_PLAYER* && player.isAlive()) {  
 allFinished = false;  
 break;  
 }  
 }  
  
 if (allFinished || board.isBoardFull()) {  
 board.announceWinner(log);  
 }  
 }  
  
 private String getColorName() {  
 Color color = GameSettings.*PLAYER\_COLORS*[playerId];  
 if (color.equals(Color.*RED*)) return "Красный";  
 if (color.equals(Color.*BLUE*)) return "Синий";  
 if (color.equals(Color.*GREEN*)) return "Зеленый";  
 if (color.equals(Color.*YELLOW*)) return "Желтый";  
 return "Неизвестный";  
 }  
}

**Вывод:** изучены основы создания и управления потоками в Java. Создано оконное приложение игры Лото, в котором игроками являются потоки.