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 №4**

**Дисциплина: Кросс-платформенное программирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.Д. Воробьев

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Т.А. Приходько

**Тема работы:** Разработка графического приложения игры «Шахматы»

**Постановка задачи:**

Реализовать движок и интерфейс для игры в традиционные шахматы

**Описание решения:**

1. Инициализация игрового поля: Создаем игровое поле, представленное двумерным массивом grid, где каждая ячейка содержит фигуру, которая туда добавлена чтением состояния игрового движка.
2. Управление игрой: Реализуем два текстовых поля и кнопку для хода.
3. Начальное состояние задаётся стандартной конфигурацией в разработанном шахматном движке движке
4. Ввод хода триггерит слушатель, который выполняет ход, используя api движка, а затем рендерит поле заново на основе состояния движка
5. Движок реализует стандартные игровые правила шахмат, эмулирует игровое поле, игроков. Предоставляет метод makeMove как api для осуществления хода.

На рисунке 1 представлена UML-диаграмма классов движка.

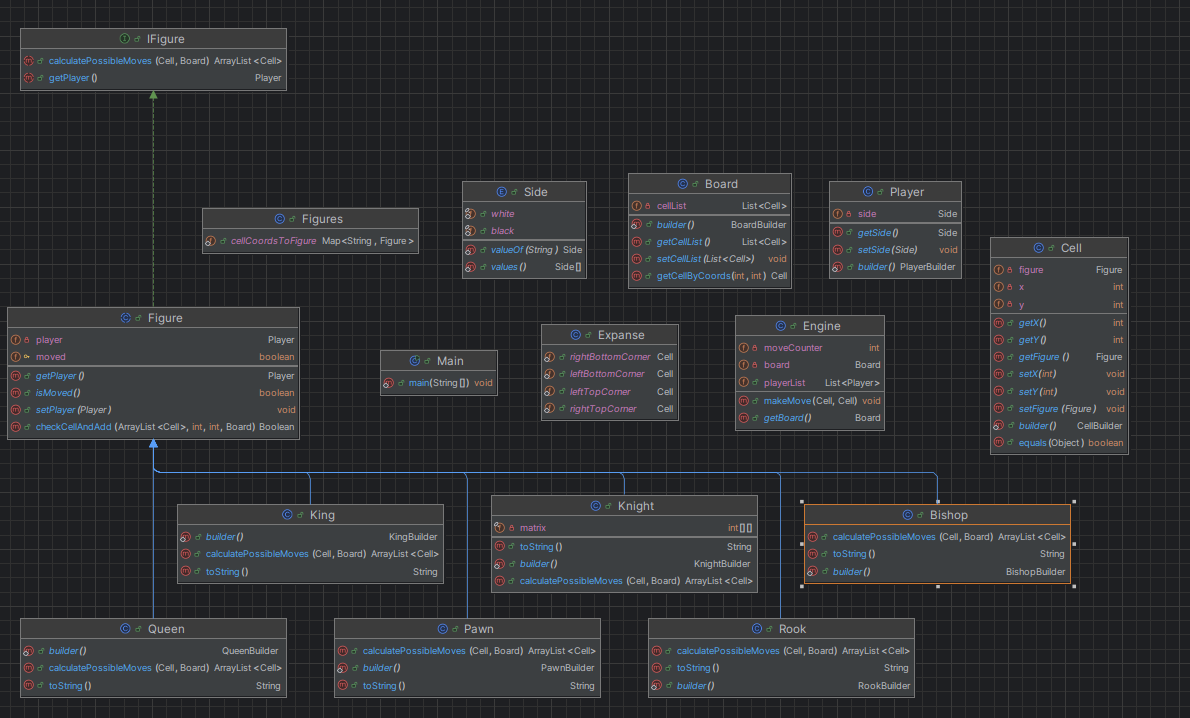
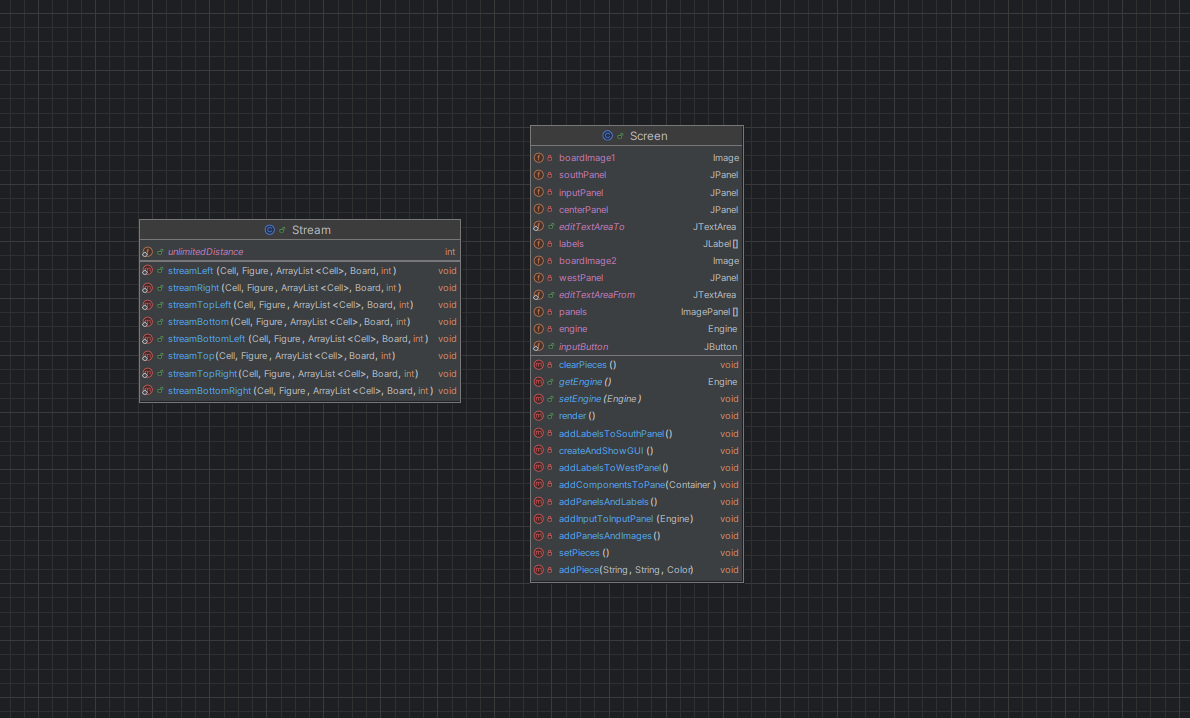


Рисунок 1 – UML-диаграмма классов движка



Также на рисунке 2 представлены диаграмма классов интерфейса

На рисунке 3 – 4 представлено выполненииe программы.

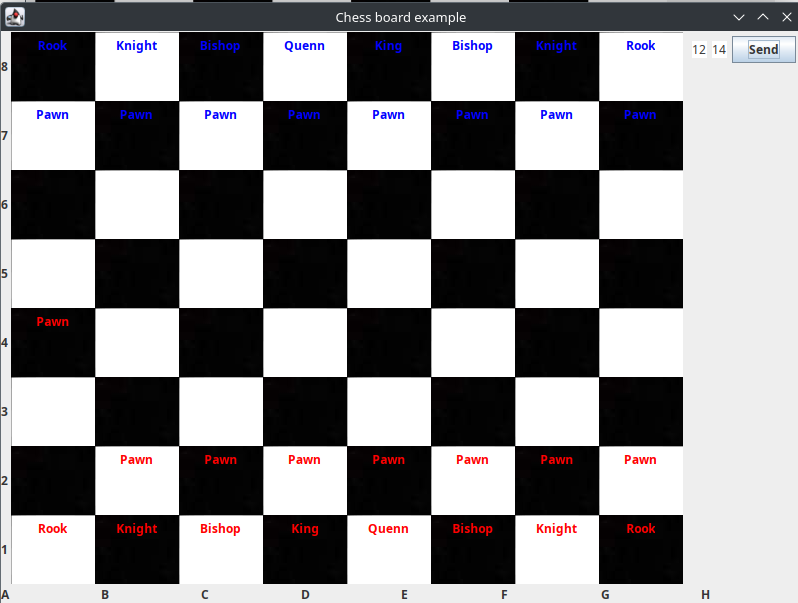


Рисунок 3 – Пример 1.

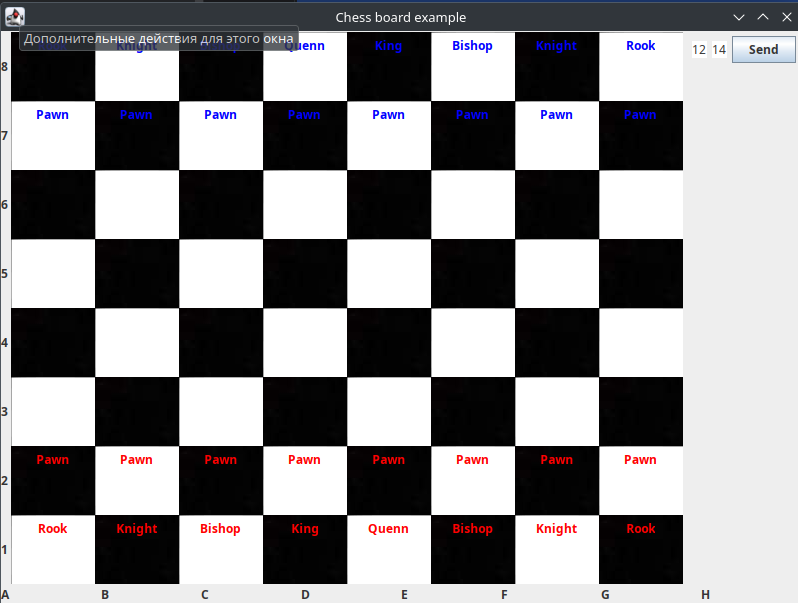


Рисунок 4 – Пример 2.

**Листинг программы**

Файл Screen.java

public class Screen extends JFrame {  
  
 //intialize variables  
 private Image boardImage1;  
 private Image boardImage2;  
 //intialize components  
 private JPanel centerPanel = new JPanel();  
 private JPanel southPanel = new JPanel();  
  
 private JPanel inputPanel = new JPanel();  
 private JPanel westPanel = new JPanel();  
 //initialze arrays to hold panels and images of the board  
 private JLabel[] labels = new JLabel[64];  
 private ImagePanel[] panels = new ImagePanel[64];  
  
 @Getter  
 @Setter  
 private Engine engine;  
  
 public Screen(Engine engine) throws IOException {  
 this.boardImage1 = ImageIO.*read*(new File("src/screen/image/block/blackBlock.png"));  
  
 this.boardImage2 = ImageIO.*read*(new File("src/screen/image/block/whiteBlock.png"));  
 this.engine = engine;  
 createAndShowGUI();//call method to create gui  
 this.addInputToInputPanel(engine);  
 this.render();  
 }  
  
 private void createAndShowGUI() {  
 setTitle("Chess board example");  
  
 setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);  
  
 addComponentsToPane(getContentPane());  
  
 setSize(800, 600);  
 setLocationRelativeTo(null);  
 setVisible(true);  
 }  
  
 */\*\*  
 \* Adds all the necessary components to the content pane of the JFrame, and  
 \* adds appropriate listeners to components.  
 \*/* private void addComponentsToPane(Container contentPane) {  
  
 GridLayout gridLayout = new GridLayout(8, 8);  
 centerPanel.setLayout(gridLayout);  
  
 //call mehod to add labels to south panel  
 addLabelsToSouthPanel();  
 //call method to add oanels to west panel  
 addLabelsToWestPanel();  
 //call method to add panels and labels to the center panel which holds the board  
 addPanelsAndLabels();  
 //add all panels to frame  
  
 contentPane.add(centerPanel, BorderLayout.*CENTER*);  
 contentPane.add(southPanel, BorderLayout.*SOUTH*);  
 contentPane.add(inputPanel, BorderLayout.*EAST*);  
 contentPane.add(westPanel, BorderLayout.*WEST*);  
 }  
  
 public static JButton *inputButton* = new JButton("Send");  
 public static JTextArea *editTextAreaFrom* = new JTextArea("12");  
 public static JTextArea *editTextAreaTo* = new JTextArea("14");  
  
  
 private void addInputToInputPanel(Engine engine) {  
 *editTextAreaFrom*.setMaximumSize(new Dimension(60,50));  
 *editTextAreaFrom*.setMinimumSize(new Dimension(60,50));  
 *editTextAreaTo*.setMaximumSize(new Dimension(60,50));  
 *editTextAreaTo*.setMinimumSize(new Dimension(60,50));  
 inputPanel.add(*editTextAreaFrom*);  
 inputPanel.add(*editTextAreaTo*);  
 inputPanel.add(*inputButton*);  
  
 *inputButton*.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 // MA - Using the class field myString to receive text from text area  
  
 String fromString = *editTextAreaFrom*.getText();  
 String toString = *editTextAreaTo*.getText();  
  
 System.*out*.println(fromString);  
 System.*out*.println(toString);  
  
 System.*out*.println(fromString.charAt(0));  
 System.*out*.println(toString);  
  
 try {  
 engine.makeMove(  
 Cell.*builder*()  
 .x(fromString.charAt(0) - '0')  
 .y(fromString.charAt(1) - '0')  
 .build(),  
 Cell.*builder*()  
 .x(toString.charAt(0) - '0')  
 .y(toString.charAt(1) - '0')  
 .build()  
 );  
 render();  
 } catch (Exception ex) {  
 System.*out*.println(ex.toString());  
 }  
  
  
// editTextAreaFrom.setText("");  
// editTextAreaTo.setText("");  
 }  
 });  
 }  
  
 private void addLabelsToSouthPanel() {  
 GridLayout gridLayout = new GridLayout(0, 8);  
  
 southPanel.setLayout(gridLayout);  
 JLabel[] lbls = new JLabel[8];  
 String[] label = {"A", "B", "C", "D", "E", "F", "G", "H"};  
  
 for (int i = 0; i < 8; i++) {  
 lbls[i] = new JLabel(label[i] + "");  
 southPanel.add(lbls[i]);  
 }  
 }  
  
 private void addLabelsToWestPanel() {  
 GridLayout gridLayout = new GridLayout(8, 0);  
  
 westPanel.setLayout(gridLayout);  
 JLabel[] lbls = new JLabel[8];  
 int[] num = {8, 7, 6, 5, 4, 3, 2, 1};  
 for (int i = 0; i < 8; i++) {  
 lbls[i] = new JLabel(num[i] + "");  
 westPanel.add(lbls[i]);  
 }  
 }  
  
  
 private void addPanelsAndLabels() {  
  
 //call methd to create panels with backgound images and appropriate names  
 addPanelsAndImages();  
  
 for (int i = 0; i < panels.length; i++) {  
 labels[i] = new JLabel();  
  
 //used to know the postion of the label on the board  
 labels[i].setName(panels[i].getName());  
  
 panels[i].add(labels[i]);  
  
 //adds panels created in addPanelsAndImages()  
 centerPanel.add(panels[i]);  
 }  
 }  
  
 //this method will create panels with backround images of chess board and set its name according to 1-8 for rows and A-H for coloumns  
 private void addPanelsAndImages() {  
 int count = 0;  
 String[] label = {"A", "B", "C", "D", "E", "F", "G", "H"};  
 int[] num = {8, 7, 6, 5, 4, 3, 2, 1};  
  
 for (int row = 0; row < 8; row++) {  
 for (int col = 0; col < 8; col++) {  
 if ((col + row) % 2 == 0) {//even numbers get white pieces  
 panels[count] = new ImagePanel(boardImage1);  
 } else {//odd numbers get black pieces  
 panels[count] = new ImagePanel(boardImage2);  
 }  
  
 panels[count].setName(label[col] + num[row]);  
 count++;  
 }  
 }  
 }  
  
 //method sets image of a label at a certain position in the board according to the block name i.e D4  
 private void addPiece(String text, String block, Color color) {  
 for (int s = 0; s < labels.length; s++) {  
 if (labels[s].getName().equalsIgnoreCase(block)) {  
 labels[s].setText(text);  
 labels[s].setForeground(color);  
 }  
 }  
 }  
  
 private void clearPieces() {  
 for (int s = 0; s < labels.length; s++) {  
 labels[s].setText("");  
 }  
 }  
  
 private void setPieces() {  
 Board board = this.engine.getBoard();  
  
 String[] label = {"A", "B", "C", "D", "E", "F", "G", "H"};  
 int[] num = {8, 7, 6, 5, 4, 3, 2, 1};  
  
 board.getCellList().forEach((item) -> {  
 if (item.getFigure() == null) {  
 return;  
 }  
  
 if (item.getFigure().getPlayer().getSide() == Side.*black*) {  
 this.addPiece(  
 item.getFigure().toString(),  
 label[item.getX() - 1] + num[8 - item.getY()],  
 Color.*BLUE* );  
 } else {  
 this.addPiece(  
 item.getFigure().toString(),  
 label[item.getX() - 1] + num[8 - item.getY()],  
 Color.*RED* );  
 }  
 });  
 }  
  
 public void render() {  
 this.clearPieces();  
 this.setPieces();  
 }  
  
 //nested class used to set the background of frame contenPane  
 class ImagePanel extends JPanel {  
  
 private Image image;  
  
 */\*\*  
 \* Default constructor used to set the image for the background for the  
 \* instance  
 \*/* public ImagePanel(Image img) {  
 image = img;  
 }  
  
 @Override  
 protected void paintComponent(Graphics g) {  
 //draws image to background to scale of frame  
 g.drawImage(image, 0, 0, null);  
 }  
 }  
}

Файл Stream.java

public class Stream {  
 public static int *unlimitedDistance* = 1000;  
 public static void streamBottomLeft(  
 Cell currentCell,  
 Figure figure,  
 ArrayList<Cell> cellArrayList,  
 Board board,  
 int distance  
 ) {  
 for (  
 int x = currentCell.getX() - 1,  
 y = currentCell.getY() - 1,  
 counter = 0;  
 x >= Expanse.*leftBottomCorner*.getX() && y >= Expanse.*leftBottomCorner*.getY() && counter < distance;  
 x--, y--, counter++  
 ) {  
 if (figure.checkCellAndAdd(cellArrayList, x, y, board)) {  
 break;  
 }  
 }  
 }  
  
 public static void streamBottom(  
 Cell currentCell,  
 Figure figure,  
 ArrayList<Cell> cellArrayList,  
 Board board,  
 int distance  
 ) {  
 for (  
 int x = currentCell.getX(),  
 y = currentCell.getY() - 1,  
 counter = 0;  
 x >= Expanse.*leftBottomCorner*.getX() && y >= Expanse.*leftBottomCorner*.getY() && counter < distance;  
 y--, counter++  
 ) {  
 if (figure.checkCellAndAdd(cellArrayList, x, y, board)) {  
 break;  
 }  
 }  
 }  
  
 public static void streamLeft(  
 Cell currentCell,  
 Figure figure,  
 ArrayList<Cell> cellArrayList,  
 Board board,  
 int distance  
 ) {  
 for (  
 int x = currentCell.getX() - 1,  
 y = currentCell.getY(),  
 counter = 0;  
 x >= Expanse.*leftBottomCorner*.getX() && y >= Expanse.*leftBottomCorner*.getY() && counter < distance;  
 x--, counter++  
 ) {  
 if (figure.checkCellAndAdd(cellArrayList, x, y, board)) {  
 break;  
 }  
 }  
 }  
  
 public static void streamTopRight(  
 Cell currentCell,  
 Figure figure,  
 ArrayList<Cell> cellArrayList,  
 Board board,  
 int distance  
 ) {  
 for (  
 int x = currentCell.getX() + 1,  
 y = currentCell.getY() + 1,  
 counter = 0;  
 x <= Expanse.*rightTopCorner*.getX() && y <= Expanse.*rightBottomCorner*.getY() && counter < distance;  
 x++, y++, counter++  
 ) {  
 if (figure.checkCellAndAdd(cellArrayList, x, y, board)) {  
 break;  
 }  
 }  
 }  
  
 public static void streamTopLeft(  
 Cell currentCell,  
 Figure figure,  
 ArrayList<Cell> cellArrayList,  
 Board board,  
 int distance  
 ) {  
 for (  
 int x = currentCell.getX() - 1,  
 y = currentCell.getY() + 1,  
 counter = 0;  
 x >= Expanse.*rightTopCorner*.getX() && y <= Expanse.*rightBottomCorner*.getY() && counter < distance;  
 x--, y++, counter++  
 ) {  
 if (figure.checkCellAndAdd(cellArrayList, x, y, board)) {  
 break;  
 }  
 }  
 }  
  
 public static void streamTop(  
 Cell currentCell,  
 Figure figure,  
 ArrayList<Cell> cellArrayList,  
 Board board,  
 int distance  
 ) {  
 for (  
 int x = currentCell.getX(),  
 y = currentCell.getY() + 1,  
 counter = 0;  
 x <= Expanse.*rightTopCorner*.getX() && y <= Expanse.*rightBottomCorner*.getY() && counter < distance;  
 y++, counter++  
 ) {  
 if (figure.checkCellAndAdd(cellArrayList, x, y, board)) {  
 break;  
 }  
 }  
 }  
  
 public static void streamRight(  
 Cell currentCell,  
 Figure figure,  
 ArrayList<Cell> cellArrayList,  
 Board board,  
 int distance  
 ) {  
 for (  
 int x = currentCell.getX() + 1,  
 y = currentCell.getY(),  
 counter = 0;  
 x <= Expanse.*rightTopCorner*.getX() && y <= Expanse.*rightBottomCorner*.getY() && counter < distance;  
 x++, counter++  
 ) {  
 if (figure.checkCellAndAdd(cellArrayList, x, y, board)) {  
 break;  
 }  
 }  
 }  
  
 public static void streamBottomRight(  
 Cell currentCell,  
 Figure figure,  
 ArrayList<Cell> cellArrayList,  
 Board board,  
 int distance  
 ) {  
 for (  
 int x = currentCell.getX() + 1,  
 y = currentCell.getY() - 1,  
 counter = 0;  
 x <= Expanse.*rightBottomCorner*.getX() && y >= Expanse.*rightBottomCorner*.getY() && counter < distance;  
 x++, y--, counter++  
 ) {  
 if (figure.checkCellAndAdd(cellArrayList, x, y, board)) {  
 break;  
 }  
 }  
 }  
}

Файл Figures.java

public class Figures {  
 public static Map<String, Figure> *cellCoordsToFigure* = new HashMap<String, Figure>() {{  
 put("12", Pawn.*builder*().build());  
 put("22", Pawn.*builder*().build());  
 put("32", Pawn.*builder*().build());  
 put("42", Pawn.*builder*().build());  
 put("52", Pawn.*builder*().build());  
 put("62", Pawn.*builder*().build());  
 put("72", Pawn.*builder*().build());  
 put("82", Pawn.*builder*().build());  
  
 put("17", Pawn.*builder*().build());  
 put("27", Pawn.*builder*().build());  
 put("37", Pawn.*builder*().build());  
 put("47", Pawn.*builder*().build());  
 put("57", Pawn.*builder*().build());  
 put("67", Pawn.*builder*().build());  
 put("77", Pawn.*builder*().build());  
 put("87", Pawn.*builder*().build());  
  
 put("11", Rook.*builder*().build());  
 put("21", Knight.*builder*().build());  
 put("31", Bishop.*builder*().build());  
 put("41", King.*builder*().build());  
 put("51", Queen.*builder*().build());  
 put("61", Bishop.*builder*().build());  
 put("71", Knight.*builder*().build());  
 put("81", Rook.*builder*().build());  
  
 put("18", Rook.*builder*().build());  
 put("28", Knight.*builder*().build());  
 put("38", Bishop.*builder*().build());  
 put("48", Queen.*builder*().build());  
 put("58", King.*builder*().build());  
 put("68", Bishop.*builder*().build());  
 put("78", Knight.*builder*().build());  
 put("88", Rook.*builder*().build());  
 }};  
}

Файл Expanse.java

public class Expanse {  
 public static Cell *leftBottomCorner* = Cell  
 .*builder*()  
 .x(1)  
 .y(1)  
 .build();  
  
 public static Cell *rightBottomCorner* = Cell  
 .*builder*()  
 .x(1)  
 .y(8)  
 .build();  
  
 public static Cell *leftTopCorner* = Cell  
 .*builder*()  
 .x(8)  
 .y(1)  
 .build();  
  
 public static Cell *rightTopCorner* = Cell  
 .*builder*()  
 .x(8)  
 .y(8)  
 .build();  
}

Файл Side.java

public enum Side {  
 *white*,  
 *black*}

Файл Player.java

@Builder  
@Getter  
@Setter  
public class Player {  
 private Side side;  
}

Файл Ifigure.java

public interface IFigure {  
 public ArrayList<Cell> calculatePossibleMoves(Cell currentCell, Board board);  
 public Player getPlayer();  
}

Файл Figure.java

public abstract class Figure implements IFigure {  
 @Getter  
 @Setter  
 private Player player;  
  
 @Getter  
 protected boolean moved = false;  
  
 public Boolean checkCellAndAdd(  
 ArrayList<Cell> cellArrayList,  
 int x,  
 int y,  
 Board board  
 ) {  
 Cell cellFromBoard = board.getCellByCoords(x, y);  
  
 if (cellFromBoard.getFigure() == null) {  
 cellArrayList.add(  
 Cell.*builder*()  
 .x(x)  
 .y(y)  
 .build()  
 );  
 return false;  
 }  
  
 if (cellFromBoard.getFigure().player.getSide() == player.getSide()) {  
 return true;  
 }  
  
 cellArrayList.add(  
 Cell.*builder*()  
 .x(x)  
 .y(y)  
 .build()  
 );  
 return true;  
 }  
}

Файл Rook.java

@Builder  
public class Rook extends Figure {  
 @Override  
 public ArrayList<Cell> calculatePossibleMoves(Cell currentCell, Board board) {  
 ArrayList<Cell> cellArrayList = new ArrayList<>();  
  
 Stream.*streamTop*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamLeft*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamRight*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamBottom*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
  
 return cellArrayList;  
 }  
  
 @Override  
 public String toString() {  
 return "Rook";  
 }  
}

Файл Quenn.java

@Builder  
public class Queen extends Figure {  
 @Override  
 public ArrayList<Cell> calculatePossibleMoves(Cell currentCell, Board board) {  
 ArrayList<Cell> cellArrayList = new ArrayList<>();  
  
 Stream.*streamBottomLeft*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamTopRight*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamTopLeft*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamBottomRight*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamTop*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamLeft*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamRight*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamBottom*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
  
 return cellArrayList;  
 }  
  
 @Override  
 public String toString() {  
 return "Quenn";  
 }  
  
}

Файл Pawn.java

@Builder  
public class Pawn extends Figure {  
 @Override  
 public ArrayList<Cell> calculatePossibleMoves(Cell currentCell, Board board) {  
 ArrayList<Cell> cellArrayList = new ArrayList<>();  
  
 if (this.getPlayer().getSide() == Side.*white*) {  
 if (currentCell.getY() + 1 > Expanse.*rightTopCorner*.getY()) {  
 return cellArrayList;  
 }  
  
 Cell forwardCell = board.getCellByCoords(  
 currentCell.getX(),  
 currentCell.getY() + 1  
 );  
  
 if (forwardCell.getFigure() == null) {  
 cellArrayList.add(forwardCell);  
 }  
  
 if (  
 currentCell.getX() - 1 >= Expanse.*leftTopCorner*.getX()  
 ) {  
 Cell nextCell = board.getCellByCoords(  
 currentCell.getX() - 1,  
 currentCell.getY() + 1  
 );  
 if (nextCell.getFigure() != null) {  
 cellArrayList.add(nextCell);  
 }  
 }  
  
 if (currentCell.getX() + 1 <= Expanse.*rightTopCorner*.getX()) {  
 Cell nextCell = board.getCellByCoords(  
 currentCell.getX() + 1,  
 currentCell.getY() + 1  
 );  
 if (nextCell.getFigure() != null) {  
 cellArrayList.add(nextCell);  
 }  
 }  
  
 if (!this.moved) {  
 if (currentCell.getY() + 2 > Expanse.*rightTopCorner*.getY()) {  
 return cellArrayList;  
 }  
  
 Cell forwardCell2 = board.getCellByCoords(  
 currentCell.getX(),  
 currentCell.getY() + 2  
 );  
  
 if (forwardCell2.getFigure() == null) {  
 cellArrayList.add(forwardCell2);  
 }  
 }  
 }  
 else {  
 if (currentCell.getY() - 1 < Expanse.*leftBottomCorner*.getY()) {  
 return cellArrayList;  
 }  
  
 Cell forwardCell = board.getCellByCoords(  
 currentCell.getX(),  
 currentCell.getY() - 1  
 );  
  
 if (forwardCell.getFigure() == null) {  
 cellArrayList.add(forwardCell);  
 }  
  
 if (  
 currentCell.getX() - 1 >= Expanse.*leftTopCorner*.getX()  
 ) {  
 Cell nextCell = board.getCellByCoords(  
 currentCell.getX() - 1,  
 currentCell.getY() - 1  
 );  
 if (nextCell.getFigure() != null) {  
 cellArrayList.add(nextCell);  
 }  
 }  
  
 if (currentCell.getX() + 1 <= Expanse.*rightTopCorner*.getX()) {  
 Cell nextCell = board.getCellByCoords(  
 currentCell.getX() + 1,  
 currentCell.getY() - 1  
 );  
 if (nextCell.getFigure() != null) {  
 cellArrayList.add(nextCell);  
 }  
 }  
  
 if (!this.moved) {  
 if (currentCell.getY() - 2 < Expanse.*leftBottomCorner*.getY()) {  
 return cellArrayList;  
 }  
  
 Cell forwardCell2 = board.getCellByCoords(  
 currentCell.getX(),  
 currentCell.getY() - 2  
 );  
  
 if (forwardCell2.getFigure() == null) {  
 cellArrayList.add(forwardCell2);  
 }  
 }  
 }  
  
 return cellArrayList;  
 }  
  
 @Override  
 public String toString() {  
 return "Pawn";  
 }  
}

Файл Knight.java

@Builder  
public class Knight extends Figure {  
 private final int[][] matrix = {  
 {2, 1},  
 {2, -1},  
 {-2, 1},  
 {-2, -1},  
 {1, 2},  
 {-1, 2},  
 {1, -2},  
 {-1, -2},  
 };  
  
 @Override  
 public ArrayList<Cell> calculatePossibleMoves(Cell currentCell, Board board) {  
 ArrayList<Cell> cellArrayList = new ArrayList<>();  
  
 Arrays.*stream*(matrix).forEach((item) -> {  
 try {  
 Cell nextCell = board.getCellByCoords(  
 currentCell.getX() + item[0],  
 currentCell.getY() + item[1]  
 );  
  
 super.checkCellAndAdd(cellArrayList,nextCell.getX(),nextCell.getY(), board);  
 } catch (Exception e) {  
 System.*out*.println(e);  
 }  
 });  
  
  
 return cellArrayList;  
 }  
  
 @Override  
 public String toString() {  
 return "Knight";  
 }  
}

Файл King.java

@Builder  
public class King extends Figure {  
 @Override  
 public ArrayList<Cell> calculatePossibleMoves(Cell currentCell, Board board) {  
 ArrayList<Cell> cellArrayList = new ArrayList<>();  
  
 Stream.*streamBottomLeft*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 1  
 );  
 Stream.*streamTopRight*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 1  
 );  
 Stream.*streamTopLeft*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 1  
 );  
 Stream.*streamBottomRight*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 1  
 );  
 Stream.*streamTop*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 1  
 );  
 Stream.*streamLeft*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 1  
 );  
 Stream.*streamRight*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 1  
 );  
 Stream.*streamBottom*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 1  
 );  
  
 return cellArrayList;  
 }  
  
 @Override  
 public String toString() {  
 return "King";  
 }  
}

Файл Bishop.java

Builder  
public class Bishop extends Figure {  
 @Override  
 public ArrayList<Cell> calculatePossibleMoves(Cell currentCell, Board board) {  
 ArrayList<Cell> cellArrayList = new ArrayList<>();  
  
 Stream.*streamBottomLeft*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamTopRight*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamTopLeft*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
 Stream.*streamBottomRight*(  
 currentCell,  
 this,  
 cellArrayList,  
 board,  
 *unlimitedDistance* );  
  
 return cellArrayList;  
 }  
  
 @Override  
 public String toString() {  
 return "Bishop";  
 }  
}

Файл Java.java

public class Engine {  
 @Getter  
 private Board board;  
 public List<Player> playerList;  
  
 private int moveCounter;  
  
 public Engine() {  
 this.moveCounter = 0;  
 this.playerList = new ArrayList<>();  
 this.playerList.add(  
 Player.*builder*()  
 .side(Side.*black*)  
 .build()  
 );  
 this.playerList.add(  
 Player.*builder*()  
 .side(Side.*white*)  
 .build()  
 );  
  
 List<Cell> cellList = new ArrayList<>();  
  
 for (int i = 1; i <= 8; i++) {  
 for (int j = 1; j <= 8; j++) {  
 String key = String.*valueOf*(i) + String.*valueOf*(j);  
 Figure figure = Figures.*cellCoordsToFigure*.get(key);  
 Cell cell = Cell.*builder*().x(i).y(j).build();  
 if (figure != null) {  
  
 if (j > 4) {  
 figure.setPlayer(this.playerList.get(0));  
 } else {  
 figure.setPlayer(this.playerList.get(1));  
 }  
  
 cell.setFigure(figure);  
 }  
 cellList.add(cell);  
 }  
 }  
  
 this.board = Board.*builder*()  
 .cellList(cellList)  
 .build();  
  
 }  
  
 public void makeMove(Cell fromCell, Cell toCell) throws Exception {  
 Cell boardFromCell = this.board.getCellByCoords(fromCell.getX(), fromCell.getY());  
  
 if (boardFromCell.getFigure() == null) {  
 throw new Exception("No figure in this cell");  
 }  
  
 if (moveCounter % 2 == 0) {  
 if (boardFromCell.getFigure().getPlayer().getSide() == Side.*black*) {  
 throw new Exception("Move not allowed, wrong side");  
 }  
 } else {  
 if (boardFromCell.getFigure().getPlayer().getSide() == Side.*white*) {  
 throw new Exception("Move not allowed, wrong side");  
 }  
 }  
  
 System.*out*.println(boardFromCell.getFigure());  
 System.*out*.println(boardFromCell.getFigure().getPlayer().getSide());  
  
 List<Cell> allowedMoves = boardFromCell.getFigure().calculatePossibleMoves(boardFromCell, this.board);  
  
 allowedMoves.forEach((item) -> {  
 System.*out*.printf("X:%d Y:%d Figure:%s\n",item.getX(), item.getY(), item.getFigure());  
 });  
  
 if (allowedMoves.contains(toCell)) {  
 System.*out*.println("Move allowed");  
 }  
 else {  
 throw new Exception("Move not allowed");  
 }  
  
 Cell boardToCell = this.board.getCellByCoords(toCell.getX(), toCell.getY());  
  
 System.*out*.printf(  
 "Move X:%d Y:%d Figure:%s \n-> X:%d Y:%d Figure:%s\n",  
 boardFromCell.getX(),  
 boardFromCell.getY(),  
 boardFromCell.getFigure(),  
 boardToCell.getX(),  
 boardToCell.getY(),  
 boardToCell.getFigure()  
 );  
  
 boardToCell.setFigure(boardFromCell.getFigure());  
 boardFromCell.setFigure(null);  
 this.moveCounter++;  
 }  
}

Файл Board.java

@Builder  
public class Board {  
 @Getter  
 @Setter  
 private List<Cell> cellList;  
  
 public Cell getCellByCoords(int x, int y) {  
 return cellList.stream()  
 .filter((item) -> item.getX() == x && item.getY() == y)  
 .findFirst()  
 .get();  
 }  
}

Файл Cell.java

@Getter  
@Builder  
@Setter  
public class Cell {  
 private int x;  
 private int y;  
 private Figure figure;  
  
 @Override  
 public boolean equals(Object obj) {  
 if (obj == this) {  
 return true;  
 }  
  
 if (obj.getClass() != this.getClass()) {  
 return false;  
 }  
  
 Cell other = (Cell) obj;  
  
 return other.x == this.x  
 && other.y == this.y;  
 }  
}