Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Základy Objektovo-orientovaného programovania

Gamifikácia

Artúr Kozubov

Meno cvičiaceho: Bc. Roman Bitarovský

Čas cvičení: Št 12:00

Dátum vytvorenia: 16. 12. 2023

Realizácia zadania „Gamifikácia”

Našou úlohou bolo **gamifikovať nejaký proces nášho života**.

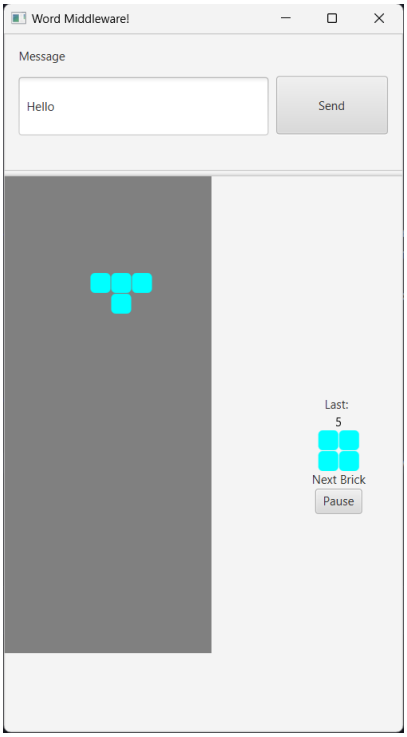
Po dlhom premýšľaní som prišiel na nápad, ako vyriešiť jeden globálny problém v komunikácii – obrovské množstvo nezmyselného textu, ktorého musia ľudia prečítať tony.

Rozhodol som sa nejakým spôsobom vytvoriť nástroj na písanie čistého a jasného textu (správy). Teda, ako povedal Joseph Sheridan, britský spisovateľ 18. a 19. storočia - *“Stručnosť je kľúčom k jasnosti.”*.

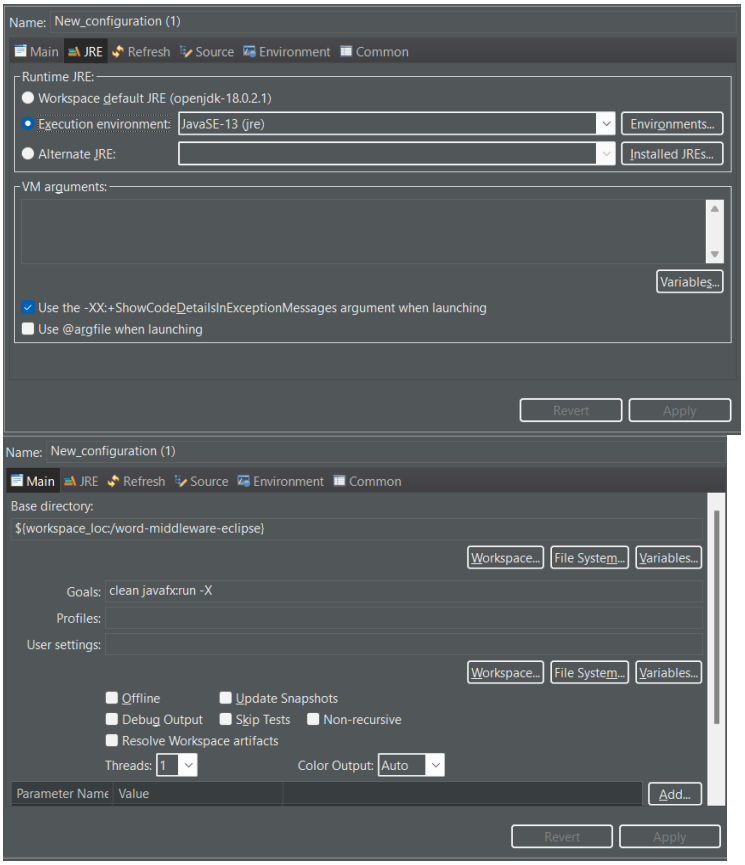
Takže na stroji vznikol program, ktorý spúšťa mini hru (Tetris) na základe náročnosti vášho textu (počtu písmen), ktorý je potrebné absolvovať, aby bolo možné poslať správu.

Vybrané technológie

- `openjdk-18.0.2.1`
- `javaFx`
  - Pre grafické rozhranie.
- `maven`
  - Nástroj na zostavovanie, ktorý uľahčuje správu závislostí a zostavovanie projektu.

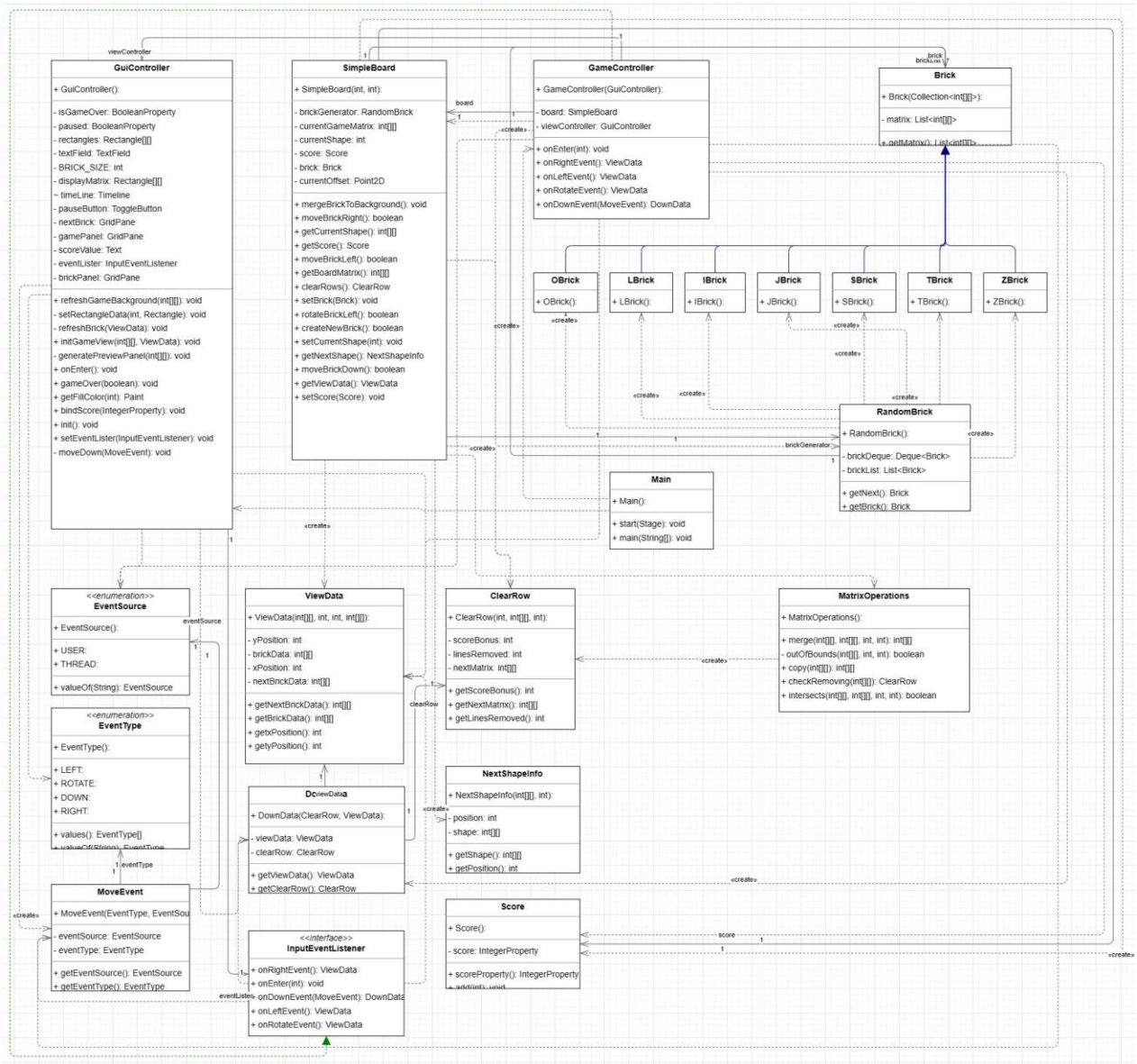


Spustenie programu (Eclipse)



pouziteOopPrincipy.pdf

UML Diagram



Nove

Default method

```
public interface InputEventListener {

  default void onDownEvent() throws ExemptionMechanismException {
    throw new ExemptionMechanismException();
  }

  //...
}
```

Abstract method

```
public abstract class Brick {
    //...

    abstract String getBrickName();
}

public class IBrick extends Brick {
    //...

    @Override
    String getBrickName() {
        return this.getClass().getName();
    }
}
```

Zmysluplné hierarchie dedenia

IBrick.java

```
public class IBrick /*- ZBrick*/ extends Brick {
    public IBrick /*- ZBrick*/() {
        super(new ArrayList<>() {{
            // ...
        }});
```

```
    }
}
```

Zapuzdrenie a správne modifikátory prístupu (členské premenné sú private)

```
public class GameController implements InputEventListener {
    private final SimpleBoard board = new SimpleBoard(25, 10);
    private final GuiController viewController;

    protected GameController(GuiController c) {
        this.viewController = c;
        this.viewController.setEventLister(this);
    }
}
```

etc... in all classes.

Použitie preťažovanie (overloading) metód

```
public class NextShapeInfo {
    public NextShapeInfo(int[][] shape) {
        this.shape = shape;
        this.position = 0;
    }

    public NextShapeInfo(int[][] shape, int position) {
        this.shape = shape;
        this.position = position;
    }
}
```

Použitie prekonávanie (overriding) metód

```
public class GameController implements InputEventListener {
    @Override
    public void onEnter(int length) {
        // ...
    }

    @Override
    public DownData onDownEvent(MoveEvent event) {
        // ...
    }

    @Override
    public ViewData onLeftEvent() {
        // ...
    }

    @Override
    public ViewData onRightEvent() {
        // ...
    }
```

```
    @Override
    public ViewData onRotateEvent() {
        // ...
    }
}

public class Main extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        // ...
    }
}

public class GuiController {
    public void init() {
        gamePanel.setOnKeyPressed(new EventHandler<KeyEvent>() {
            @Override
            public void handle(KeyEvent event) {
                if (paused.getValue() == Boolean.FALSE && isGameOver.getValue
() == Boolean.FALSE) {
                    if (event.getCode() == KeyCode.UP || event.getCode() == K
eyCode.W) {
                        refreshBrick(eventLister.onRotateEvent());
                        event.consume();
                    }
                    if (event.getCode() == KeyCode.DOWN || event.getCode() ==
KeyCode.S) {
                        moveDown(new MoveEvent(EventType.DOWN, EventSource.US
ER));
                        System.out.println("Down");
                        event.consume();
                    }
                    if (event.getCode() == KeyCode.LEFT || event.getCode() ==
KeyCode.A) {
                        refreshBrick(eventLister.onLeftEvent());
                        event.consume();
                    }
                    if (event.getCode() == KeyCode.RIGHT || event.getCode() =
= KeyCode.D) {
                        refreshBrick(eventLister.onRightEvent());
                        event.consume();
                    }
                }

                if (event.getCode() == KeyCode.P) {
                    pauseButton.selectedProperty().setValue(!pauseButton.sele
ctedProperty().getValue());
                }
            }
```

```
        });

        pauseButton.selectedProperty().addListener(new ChangeListener<Boolean
>() {
            @Override
            public void changed(ObservableValue<? extends Boolean> observable
, Boolean oldValue, Boolean newValue) {
                if (newValue) {
                    timeLine.pause();
                    pauseButton.setText("Resume");
                } else {
                    timeLine.play();
                    pauseButton.setText("Pause");
                }
            }
        });
    }
}

public abstract class Brick {
    @Override
    public int hashCode() {
        // ...
    }

    @Override
    public boolean equals(Object obj) {
        // ...
    }

    @Override
    public String toString() {
        // ...
    }
}
```

Použitie vzťahu agregácie

```
public class GuiController {
    @FXML
    private ToggleButton pauseButton;
    @FXML
    private GridPane gamePanel;
    @FXML
    private GridPane nextBrick;
    @FXML
    private GridPane brickPanel;
    @FXML
    private Text scoreValue;
```

```
    @FXML
    private TextField textField;
}

public class GameController implements InputEventListener {
    private final SimpleBoard board = new SimpleBoard(25, 10);
    private final GuiController viewController;
}

public class RandomBrick {
    private final List<Brick> brickList;
    private final Deque<Brick> brickDeque = new ArrayDeque<>();
}
```

etc…

Použitie vzťahu kompozície

```
public class GuiController {
    Timeline timeLine;

    public void initGameView(int[][] boardMatrix, ViewData viewData) {
        timeLine = new Timeline(
                new KeyFrame(Duration.millis(400), ae -> moveDown(new MoveEve
nt(EventType.DOWN,
                        EventSource.THREAD))));
        timeLine.setCycleCount(Timeline.INDEFINITE);
        timeLine.play();
    }
}
```

Použitie vzťahu asociácie

```
public class GameController implements InputEventListener {
    private final SimpleBoard board = new SimpleBoard(25, 10);
}
```

Použitie finálny atribút a metódu

```
public abstract class Brick {
    private final List<int[][]> matrix = new ArrayList<>();

    public final List<int[][]> getMatrix() {
        return matrix;
    }
}

public class GuiController {
    private static final int BRICK_SIZE = 20;
    private final BooleanProperty paused = new SimpleBooleanProperty();
    private final BooleanProperty isGameOver = new SimpleBooleanProperty();
    Timeline timeLine;
    private InputEventListener eventLister;
    private Rectangle[][] displayMatrix;
```

```
    private Rectangle[][] rectangles;

    @FXML
    private ToggleButton pauseButton;
    @FXML
    private GridPane gamePanel;
    @FXML
    private GridPane nextBrick;
    @FXML
    private GridPane brickPanel;
    @FXML
    private Text scoreValue;
    @FXML
    private TextField textField;
}
```

Použitie abstraktnej triedu

```
public abstract class Brick {
}
```

Použitie statickej metódu a atribútu

```
public class MatrixOperations {
    public static boolean intersects(int[][] matrix, int[][] brick, int x, in
t y) {
        // ...
    }

    public static int[][] merge(int[][] filledFields, int[][] brick, int x, i
nt y) {
        // ...
    }

    public static int[][] copy(int[][] original) {
        // ...
    }

    private static boolean outOfBounds(int[][] matrix, int targetX, int targe
tY) {
        // ...
    }

    public static ClearRow checkRemoving(int[][] matrix) {
        // ...
    }
}

public class GuiController {
    private static final int BRICK_SIZE = 20;
}
```

```
public class Main extends Application {
    public static void main(String[] args) {
        // ...
    }
}
```

Použitie upcasting-u

```
public class RandomBrick {
    private final List<Brick> brickList;
    private final Deque<Brick> brickDeque = new ArrayDeque<>();

    public RandomBrick() {
        brickList = Arrays.asList(new ZBrick(), new TBrick(), new SBrick(), n
ew OBrick(), new LBrick(), new JBrick(), new IBrick());

        brickDeque.add(brickList.get((int) (Math.random() * brickList.size())
));
        brickDeque.add(brickList.get((int) (Math.random() * brickList.size())
));
        brickDeque.add(brickList.get((int) (Math.random() * brickList.size())
));
        brickDeque.add(brickList.get((int) (Math.random() * brickList.size())
));
    }
}
```

JavaDoc

```
public class MatrixOperations {
    /**
     * Merges a brick with a given matrix at the specified position.
     *
     * @param filledFields the matrix to merge the brick with
     * @param brick        the brick to merge
     * @param x            the x position of the brick
     * @param y            the y position of the brick
     * @return the merged matrix with the brick incorporated
     */
    public static int[][] merge(int[][] filledFields, int[][] brick, int x, i
nt y) {
        // ...
    }
}
```