Rebeka Cocherová, Ing. Oliver Udvardi, CPUb, Utorok 16:00

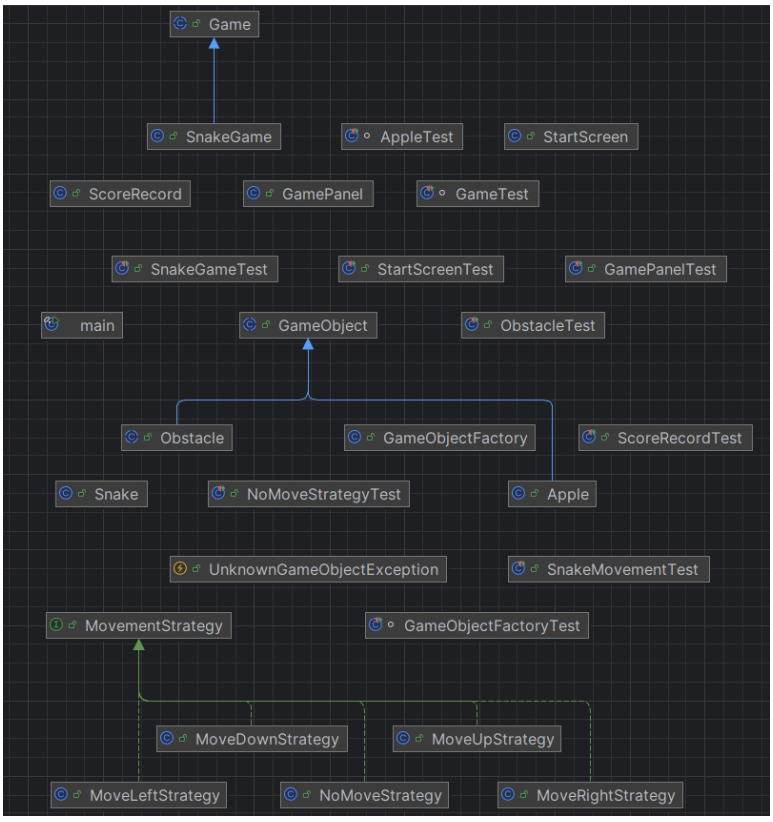
Projekt z objektovo orientovaného programovania LS2025

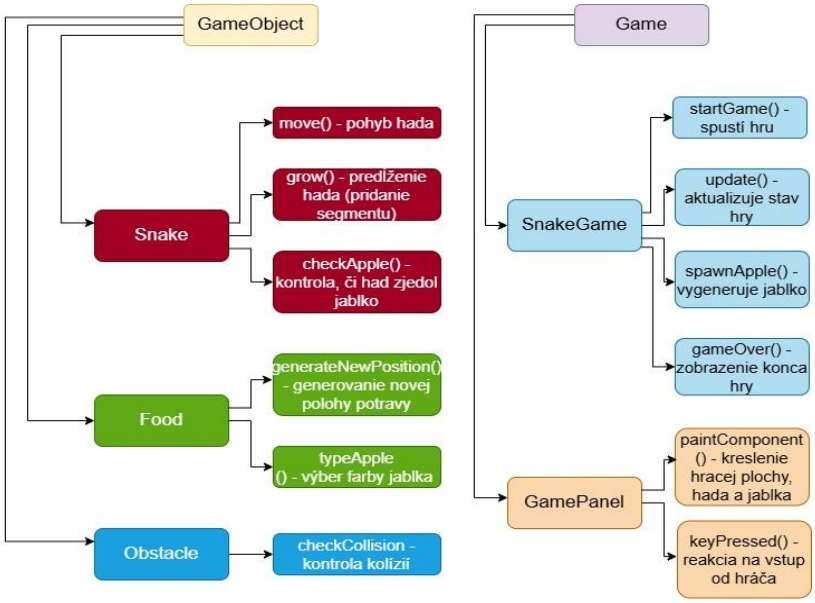
Zámer projektu

Po spustení hry v maine sa otvorí obrazovka, na ktorej si treba vybrať farbu hada a farbu potravy. Po tom, ako si používateľ vyberie farby, sa spustí hra Hadík. Otvorí sa okno, na ktorom je vykreslená zelená plocha, farebné kvietky na náhodných súradniciach a orámovanie hernej obrazovky (plot). Had sa nevykreslí v ľavom hornom rohu, ale trošku nižšie (vyzerá to esteticky lepšie). Had sa začne pohybovať po obrazovke a cieľom hráča je nazbierať čo najviac jabĺk. Tieto jablká sa generujú na náhodných pozíciách. Počas hry sa v hornom ľavom rohu počíta čas hry a skóre. PO skončení hry sa do textového súboru zapíše čas a skóre. V prípade stlačenia Enter sa hra spustí znova.

Žáner: *arkádová hra*

UML diagram tried





UML diagram zo zámeru

Použité knižnice

1. junit:junit:4.13.1

- *písanie jednotkových testov*
- *obsahuje napr. anotácie @Test, @Before, @After*
- *overovanie správnosti kódu*

2. org.apiguardian:apiguardian-api:1.1.2

- *malá knižnica s anotáciami na označenie stability API*
- *napr. či je niečo @PublicApi, @Internal*

3. org.hamcrest:hamcrest-core:1.3

- *Zjednodušuje písanie asercií (overení) v testoch*
- *namiesto assertEquals(5, value) sa môže písať assertThat(value, is(5))*
- *veľmi čitateľné a flexibilné overenia výsledkov*

4. org.jcommander:jcommander:1.83

- *parsovanie argumentov príkazového riadku (napr. --file test.txt).*
- *veľmi jednoduché mapovanie argumentov na Java objekty.*

5. org.junit.jupiter:junit-jupiter:5.8.1

- hlavné API pre JUnit 5.

- obsahuje všetko na písanie nových testov: anotácie (@Test, @BeforeEach), assertions

6. org.junit.jupiter:junit-jupiter-api:5.8.1

- API rozhranie pre JUnit 5

- definuje anotácie, assertions a pomocné triedy, ktoré sa používajú pri písaní testov

- je to časť JUnit Jupiter

7. org.junit.jupiter:junit-jupiter-engine:5.8.1

- engine, ktorý umožňuje spúšťanie JUnit 5 testov

- po napísaní a spustení testu ho práve tento engine reálne vykoná

8. org.junit.jupiter:junit-jupiter-params:5.8.1

- rozšírenie JUnit 5 na parameterized tests

- umožňuje spúšťať jeden test s rôznymi dátovými sadami (@ParameterizedTest)

9. org.junit.platform:junit-platform-commons:1.8.1

- spoločné utility pre rôzne časti JUnit Platform

- obsahuje základné triedy na správu testov, chyby

- používa sa vnútri JUnit platformy

10. org.junit.platform:junit-platform-engine:1.8.1

- definuje štandardy, ako majú vyzerať testovacie enginy

- JUnit 5 aj iné frameworky môžu pomocou tohto vytvárať vlastné enginy

11. org.opentest4j:opentest4j:1.2.0

- štandardné rozhranie pre chyby v testoch (napr. keď test zlyhá, ako to nahlásiť).

- používa sa vo viacerých testovacích frameworkoch (napr. JUnit 5).

12. org.slf4j:slf4j-api:1.7.36

- API pre logovanie v Jave

- neobsahuje konkrétnu implementáciu logovania, len definície, ako by malo logovanie fungovať

13. org.slf4j:slf4j-simple:1.7.36

- jednoduchá implementácia SLF4J

- logy sa vypisujú jednoducho do konzoly (System.out)

14. org.testng:testng:7.11.0

- alternatíva k JUnit

- framework na unit a integračné testovanie

- silnejší v podpore pre závislosti medzi testami, parameterized tests a konfigurácie

15. org.webjars:jquery:3.7.1

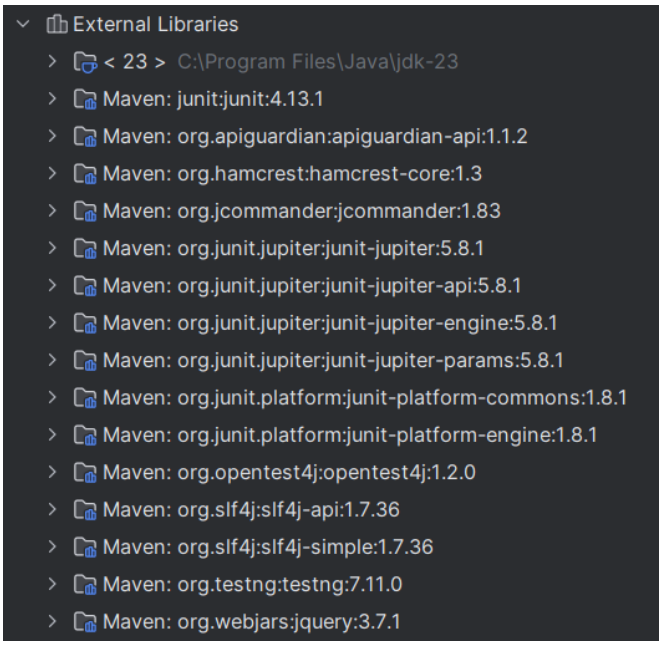
- WebJar balenie jQuery pre Maven projekty

- umožňuje jednoducho používať jQuery vo web aplikáciách bez toho, aby si musel riešiť CDN alebo manuálne kopírovanie súborov

Vyjadrenie sa k splneniu podmienok (nutnych, aj dalsich)

- dedenie a rozhrania
    - GameObject je abstraktná trieda, z ktorej dedí napríklad Apple a Obstacle
    - rozhranie MovementStrategy, ktoré implementujú triedy ako MoveUpStrategy, MoveDownStrategy, MoveLeftStrategy, MoveRightStrategy.
    - príklady v kóde:
        - public abstract class GameObject
        - public class Apple extends GameObject
        - public class Obstacle extends GameObject
        - public interface MovementStrategy
        - public class MoveLeftStrategy implements MovementStrategy
        - public class MoveRightStrategy implements MovementStrategy
        - public class MoveUpStrategy implements MovementStrategy
        - public class MoveDownStrategy implements MovementStrategy
        - public class NoMoveStrategy implements MovementStrategy

- Maven pri externých knižniciach

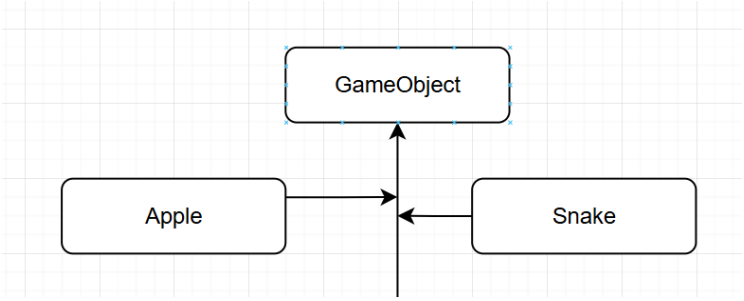


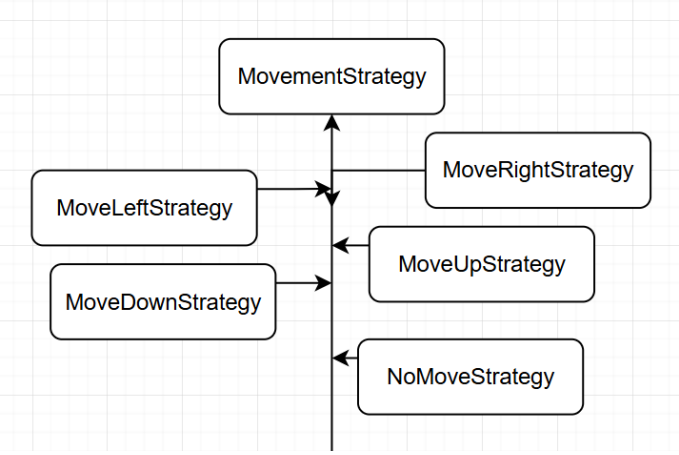
- použitie základných OOP princípov
    - abstrakcia: Game, GameObject, Obstacle sú abstraktné triedy
    - dedenie: : Apple a Obstacle dedia GameObject
    - enkapsulácia: premenné sú chránené a prístupné cez gettery
    - polymorfizmus: napr. MovementStrategy - môže meniť pohybové straté-gie za behu
    - príklady v kóde:
        - public abstract class GameObject
        - public class Apple extends GameObject
        - public class Obstacle extends GameObject
        - public interface MovementStrategy
        - @Override
          public void move(Snake snake)
        - private int score
        - private Color snakeColor

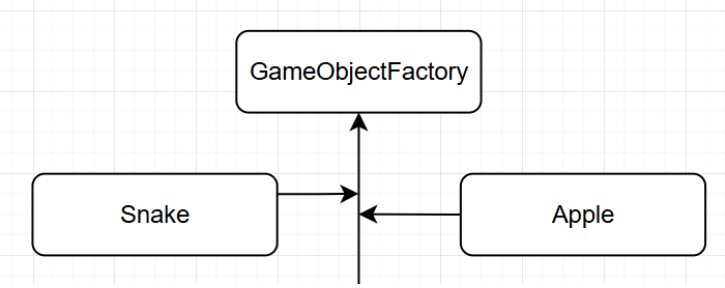
- jednotkové testy

Element	Class, %	Method, %	Line, %	Branch, %
all	94% (34/36)	94% (147/156)	95% (557/583)	67% (84/125)
Apple	100% (2/2)	100% (12/12)	100% (28/28)	100% (2/2)
AppleTest	100% (2/2)	100% (9/9)	100% (38/38)	66% (8/12)
Game	0% (0/1)	100% (0/0)	100% (0/0)	100% (0/0)
GameObject	100% (2/2)	100% (6/6)	100% (10/10)	100% (0/0)
GameObjectFactory	100% (1/1)	100% (2/2)	87% (29/33)	100% (8/8)
GameObjectFactoryTest	100% (1/1)	100% (8/8)	100% (15/15)	100% (0/0)
GamePanel	100% (2/2)	100% (8/8)	100% (86/86)	92% (12/13)
GamePanelTest	100% (1/1)	100% (5/5)	93% (15/16)	60% (6/10)
GameTest	100% (2/2)	100% (7/7)	100% (15/15)	100% (0/0)
MoveDownStrategy	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
MoveLeftStrategy	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
MovementStrategy	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
MoveRightStrategy	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
MoveUpStrategy	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
NoMoveStrategy	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
NoMoveStrategyTest	100% (1/1)	100% (2/2)	100% (11/11)	100% (0/0)
Obstacle	100% (2/2)	100% (3/3)	100% (3/3)	100% (4/4)
ObstacleTest	66% (2/3)	50% (4/8)	85% (24/28)	100% (0/0)
ScoreRecord	100% (1/1)	100% (4/4)	100% (6/6)	100% (0/0)
ScoreRecordTest	100% (1/1)	100% (4/4)	100% (16/16)	100% (0/0)
Snake	100% (2/2)	95% (19/20)	97% (45/46)	76% (20/26)
SnakeGame	100% (2/2)	90% (20/22)	84% (66/78)	40% (13/32)
SnakeGameTest	100% (1/1)	100% (10/10)	100% (71/71)	58% (7/12)
SnakeMovementTest	100% (1/1)	100% (5/5)	100% (21/21)	100% (0/0)
StartScreen	100% (1/1)	60% (3/5)	83% (20/24)	75% (3/4)
StartScreenTest	100% (1/1)	100% (10/10)	100% (32/32)	50% (1/2)
UnknownGameObjectException	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)

- použitie návrhových vzorov
  - Builder pattern: používaný v Apple.Builder, Snake.Builder, GameObject.Builder
  - Strategy pattern: MovementStrategy + implementácie (rôzne pohyby hada)
  - Factory pattern: GameObjectFactory vytvára objekty typu Snake a Apple







- logovanie základných činností (info, warning, error)
    - v GameObjectFactory sa používa Logger (slf4j)

```
public static Object create(String type, int width, int height) {  10 usages
    if (type == null || type.isBlank()) {
        logger.warn("Received blank or null type for GameObject creation.");
        throw new UnknownGameObjectException("Type cannot be null or blank");
    }
```

```
        Object gameObject = builderClass.getMethod( name: "build").invoke(builder);
        logger.info("Successfully created object of type: {}", type);
        return gameObject;

    } catch (ClassNotFoundException e) {
        logger.error("Class not found for type: {}", type, e);
        throw new UnknownGameObjectException("Class not found for type: " + type);
```

```
public class StartScreen extends JFrame {  7 usages
    private Color snakeColor;  3 usages
    private Color foodColor;  3 usages
    private JButton startButton;  5 usages

    public StartScreen() {  2 usages
        setTitle("Výber farieb");
        setSize( width: 300,  height: 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        JButton snakeColorBtn = new JButton( text: "Výber farby hada");
        JButton foodColorBtn = new JButton( text: "Výber farby potravy");
        startButton = new JButton( text: "Začať hru");
        startButton.setEnabled(false);
```

- implementácia a využitie vlastných výnimiek

```
public class UnknownGameObjectException extends RuntimeException {  10 usages
    public UnknownGameObjectException(String message) {  4 usages
        super(message);
    }
}
```

- implementácia a vytvorenie GUI
    - GamePanel
    - SnakeGame
    - StartScreen

- explicitné použitie viacvláknovosti
    - v SnakeGame je použité vlákno (Thread) na hernú slučku (star-GameThread)
    - keď sa zavolá startGame(), inicializuje sa nový had, vygeneruje nové ja-blko, resetuje sa skóre a spustí sa **nekonečný herný cyklus** v samostat-nom vlákne, ktorý pravidelne aktualizuje hru a vykresľuje nové stavy

```java
@Override 5 usages
public void startGame() {
    running = true;
    score = 0;
    currentDelay = 100;
    snake = (Snake) GameObjectFactory.create( type: "snake",  width: 600,  height: 600);
    apple.generateNewPosition();
    startTime = System.currentTimeMillis();
    elapsedTime = 0;
    startGameThread();
}
private void startGameThread() {  1 usage
    if (gameThread != null && gameThread.isAlive()) {
        gameThread.interrupt();
    }
    gameThread = new Thread(() -> {
        while (running) {
            update();
            gamePanel.repaint();
            try {
                Thread.sleep(currentDelay);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                break;
            }
        }
    });
    gameThread.start();
}
```

- použitie generickosti
  - v GameObject.Builder<T extends Builder<T>> je použitá **generika**
  - týka sa tu Builder patternu
  - príklad v kóde:
    - public static abstract class Builder<T extends Builder<T>>

- použitie reflexie
  - v GameObjectFactory - dynamické načítanie tried podľa názvu a volanie ich metód

```java
Class<?> clazz = Class.forName(className);
Class<?> builderClass = Class.forName( className: className + "$Builder");
Object builder = builderClass.getDeclaredConstructor().newInstance();
```

- použitie lambda výrazov
  - v StartScreen - lambda tu nahrádza potrebu vytvárať triedu alebo anonymnú triedu ActionListener

```
snakeColorBtn.addActionListener( ActionEvent e -> {
    snakeColor = JColorChooser.showDialog( component: this,  title: "Zvoľ farbu hada", Color.GREEN);
    checkReady();
});

foodColorBtn.addActionListener( ActionEvent e -> {
    foodColor = JColorChooser.showDialog( component: this,  title: "Zvoľ farbu potravy", Color.RED);
    checkReady();
});

startButton.addActionListener( ActionEvent e -> {
    this.setVisible(false);
    new SnakeGame(snakeColor, foodColor);
});
```

- použitie serializácie
    - v SnakeGame - v metóde gameOver() **-** uloženie objektu do súboru v binárnej podobe

```
try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "score_record.ser"))) {
    oos.writeObject(record);
```

Navod ako spustit projekt
Hra sa spúšťa v Maine.

Pouzivatelska prirucka
Po spustení hry musí hráč ovládať hada pomocou šípok a zbierať jablká. Zároveň však nesmie naraziť do steny, ani do vlastného tela.