

Zadanie 3: Použité OOP Princípy

Ondrej Bodnár, B-INFO4

1. Dedenie

V projekte som využil princípy dedenia napríklad pri rozšírení triedy Marketplace triedou HappyHourMarket. Vďaka tomuto rozšíreniu hráč získa náhodný bonus pri predávaní svojich resources. V tomto prípade som využil aj override pôvodnej metódy v parent triede Marketplace.

```
3 usages
public class HappyHourMarket extends Marketplace{

    2 usages
    @Override
    public int sellResources(Storage s, Player p) {
        int profits = 0;
        for (Resource r : s.getContent()) {
            profits += r.getProfitPerOne();
        }
        s.getContent().clear();
        p.increaseMoney(calculateBonus(profits));

        return calculateBonus(profits);
    }

    2 usages
    private int calculateBonus(int profits) { return (int) (profits + Math.round((profits/100.0) * 50.0)); }
}
```

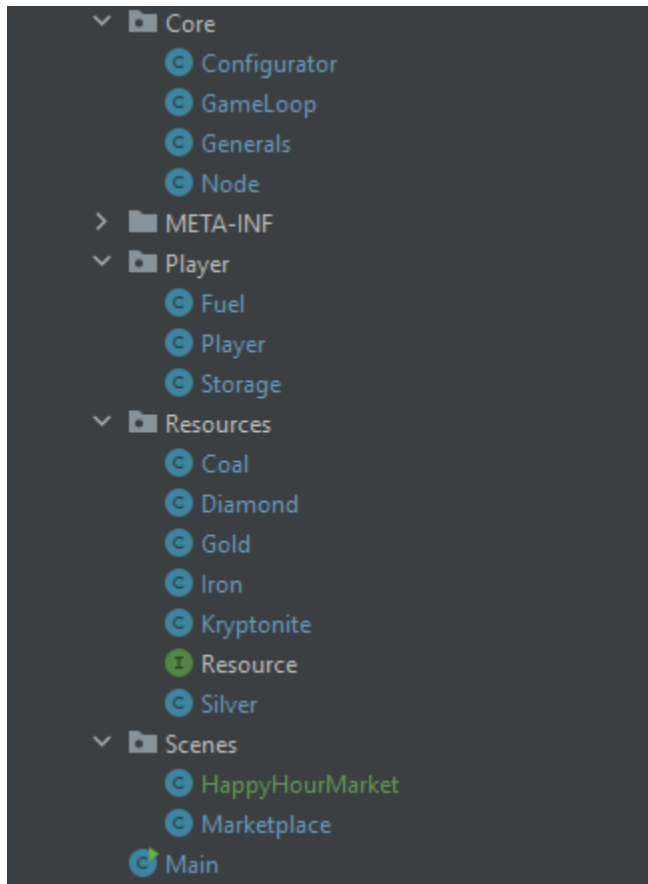
2. Zapuzdrenie

Princíp zapuzdrenia som využil v celom kóde programu. Napríklad pri triede Player, kde atribút money predstavuje senzitívnu informáciu, ktorú som chcel zabezpečiť princípom zapuzdrenia. Využitím getter a setter metód som zabezpečil, aby nastavovanie peňazí daného hráča bolo možné iba v danej triede (s pomocou private modifikátora metódy setMoney()). Následne môžeme využívať getter metódu kdekoľvek v programe, ktorá nám vráti množstvo peňazí, ktorými daný hráč disponuje.

```
FreeDayZ *
3 public class Player {
    2 usages
4     protected int money;
    1 usage
5     private final int moneyLimit = -300;
6
    2 usages
7     private int layerScore; |
8
    1 usage  FreeDayZ
9     public Player(int money) {
10         setMoney(money);
11     }
12
    3 usages  FreeDayZ
13     public boolean reachedMoneyLimit() { return getMoney() <= moneyLimit; // simplified ternary }
16
    9 usages  FreeDayZ
17     public int getMoney() { return money; }
20
    3 usages  FreeDayZ
21     private void setMoney(int money) { this.money = money; }
24
```

3. Organizácia kódu

Program je organizovaný do viacerých balíkov následovne:



4. Overloading (preťaženie) metód

Preťaženie metód som využil napríklad pri odoberaní paliva hráčovi. Postupom hry má hráč 5% šancu na získanie “zľavy” paliva. To znamená, že následujúci layer(podlažie) mu nebude odrátaná spotreba paliva (10litrov).

```
1 usage  FreeDayZ
21      public void decreaseFuel() {
22          if (fuel > 10) {
23              setFuel(fuel - fuelConsumption);
24          }
25      }
26
27      1 usage  new *
28      public void decreaseFuel(boolean isDiscounted) {
29          if (fuel > 10 && isDiscounted) {
30              setFuel(fuel);
31          }
32      }
```

5. Overriding (prekonanie) metód

Prekonávanie metód som využíval vo viacerých prípadoch. Každá trieda má prekonané metódy toString(), equals() a hashCode(). Prekonávanie som využil tiež pri nastavovaní behu programu (TimerTask), kde bolo nutné implementovať vlastný TimerTask s prekonaním default metódy run(). Okrem toho som využil prekonávanie pri vytváraní tried s implementáciou rozhrania Resource.

```

28      2 usages  FreeDayZ
29      @Override
30      public int getDropChance() { return dropChance; }
31
32
33      1 usage  FreeDayZ
34      public void setDropChance(int dropChance) { this.dropChance = dropChance; }
35
36
37      FreeDayZ
38      @Override
39      public int getProfitPerOne() {
40          return (int) Math.round((ThreadLocalRandom.current().nextDouble(
41              getMinPrice(),
42              bound: getMaxPrice() + 1.0)));
43      }

```

```

43
44      @Override
45      public String toString() {
46          return "Player{" +
47              "money=" + money +
48              ", moneyLimit=" + moneyLimit +
49              ", layerScore=" + layerScore +
50              '}';
51      }
52
53      new *
54      @Override
55      public boolean equals(Object o) {
56          if (this == o) return true;
57          if (o == null || getClass() != o.getClass()) return false;
58          Player player = (Player) o;
59          return money == player.money && layerScore == player.layerScore;
60      }
61
62      new *
63      @Override
64      public int hashCode() {
65          return Objects.hash(money, moneyLimit, layerScore);
66      }

```

6. Agregácia

Agregáciu som využil v hlavnom cykle programu. Zabezpečíme vzťah medzi triedou GameLoop a triedami Timer, Fuel, Player, Storage pomocou atribútov a enkapsulácie. Inštancie daných tried sú vytvorené v triede GameLoop iba raz.

```
5 usages  👤 FreeDayZ *  
public class GameLoop {  
    2 usages  
    public Timer t;  
    2 usages  
    public Fuel f;  
    2 usages  
    public Player p;  
  
    2 usages  
    public Storage s;
```

7. Kompozícia

Princíp kompozície bol využitý najmä v hlavnom cykle programu. Pomocou getter a setter metód som si vytvoril inštancie, ktoré sú vytvárané jedine v tomto konštruktore, nikde inde v programe. V programe k nim prístupujem pomocou getter metódy.

```
3 usages  new *  
public GameLoop(int fuelAmount, int moneyAmount, int storageAmount) {  
    setF(new Fuel(fuelAmount));  
    setP(new Player(moneyAmount));  
    setS(new Storage(storageAmount));  
}
```

10. Asociácia

Vzťah pomocou asociácie je využitý napríklad v triede Storage, kde som použil ArrayList Resource rozhraní, vďaka čomu dokážem do atribútu content vložiť rôzne objekty implementujúce Resource rozhranie a to nám slúži ako “inventár” vyťažených resources.

```
13 usages  👤 FreeDayZ *
public class Storage {

    6 usages
    private int storage;

    8 usages
    public ArrayList<Resource> content;

    1 usage  👤 FreeDayZ
    public Storage(int storage) {
        setStorage(storage);
        setContent(new ArrayList<>());
    }
}
```

11. Finálny atribút, finálna metóda

Pomocou finálneho atribútu som zabezpečil to, aby nebolo možné zmeniť obsah tohto atribútu, pomocou ktorého je kód bezpečnejší. Taktiež som využil finálne metódy, ktoré po nastavení final nemôžu byť neskôr prekonané.

Príklad finálneho atribútu moneyLimit

```
11 usages  👤 FreeDayZ *
public class Player {

    6 usages
    protected int money;

    3 usages
    private final int moneyLimit = -300;
}
```

Príklad finálnych metód


```

1 usage  👤 FreeDayZ *
public final void decreaseMoney(int price) { setMoney(getMoney() - price); }

2 usages  👤 FreeDayZ *
public final void increaseMoney(int price) { setMoney(getMoney() + price); }

```

12. Rozhranie, abstraktná trieda

V programe som využil rozhranie Resource. Toto rozhranie uľahčuje vytváranie nových resources, ktoré môže hráč počas hry objaviť. Triedy, ktoré implementujú dané rozhranie majú prekované metódy v danom rozhraní.

```

👤 FreeDayZ
public interface Resource {

    2 usages  6 implementations  👤 FreeDayZ
    int getDropChance();

    2 usages  6 implementations  👤 FreeDayZ
    int getResID();

    6 implementations  👤 FreeDayZ
    String getName();

    6 implementations  👤 FreeDayZ
    int getProfitPerOne();

    6 usages  6 implementations  👤 FreeDayZ
    double getMinPrice();

    6 usages  6 implementations  👤 FreeDayZ
    double getMaxPrice();
}

```

```

2 usages  🧑 FreeDayZ
@Override
public int getResID() { return resID; }

1 usage  🧑 FreeDayZ
public void setResID(int resID) { this.resID = resID; }

2 usages  🧑 FreeDayZ
@Override
public int getDropChance() { return dropChance; }

1 usage  🧑 FreeDayZ
public void setDropChance(int dropChance) { this.dropChance = dropChance; }

🧑 FreeDayZ
@Override
public int getProfitPerOne() {
    return (int) Math.round((ThreadLocalRandom.current().nextDouble(
        getMinPrice(),
        bound: getMaxPrice() + 1.0)));
}

```

Ukážka triedy Coal, ktorá implementuje rozhranie Resource

13. Statická metóda, statický atribút, Singleton

V programe som využil Singleton návrhový vzor pri triede Generals. Dosiahol som ho pomocou statického atribútu inštancie triedy Generals a statickej metódy getInstance(). Okrem toho je využitý privátny konštruktor. Pri vytváraní vo funkcii Main dokážeme získať inštanciu danej triedy bez jej vytvárania. Inštancia je vytvorená už v triede a my k nej pristupujeme pomocou statickej getInstance() metódy.

```

27 usages  👤 FreeDayZ *
public class Generals {

    1 usage
    private static Generals instance = new Generals();

```

Statický atribút instance

```

1 usage  👤 FreeDayZ *
public static Generals getInstance() {
    return instance;
}

```

Statická metóda getInstance()

```

27 usages  👤 FreeDayZ *
public class Generals {

    1 usage
    private static Generals instance = new Generals();
    9 usages
    private String input;
    2 usages
    private static int scene = 0;

    1 usage  new *
    private Generals() { }

    1 usage  👤 FreeDayZ *
    public static Generals getInstance() {
        return instance;
    }

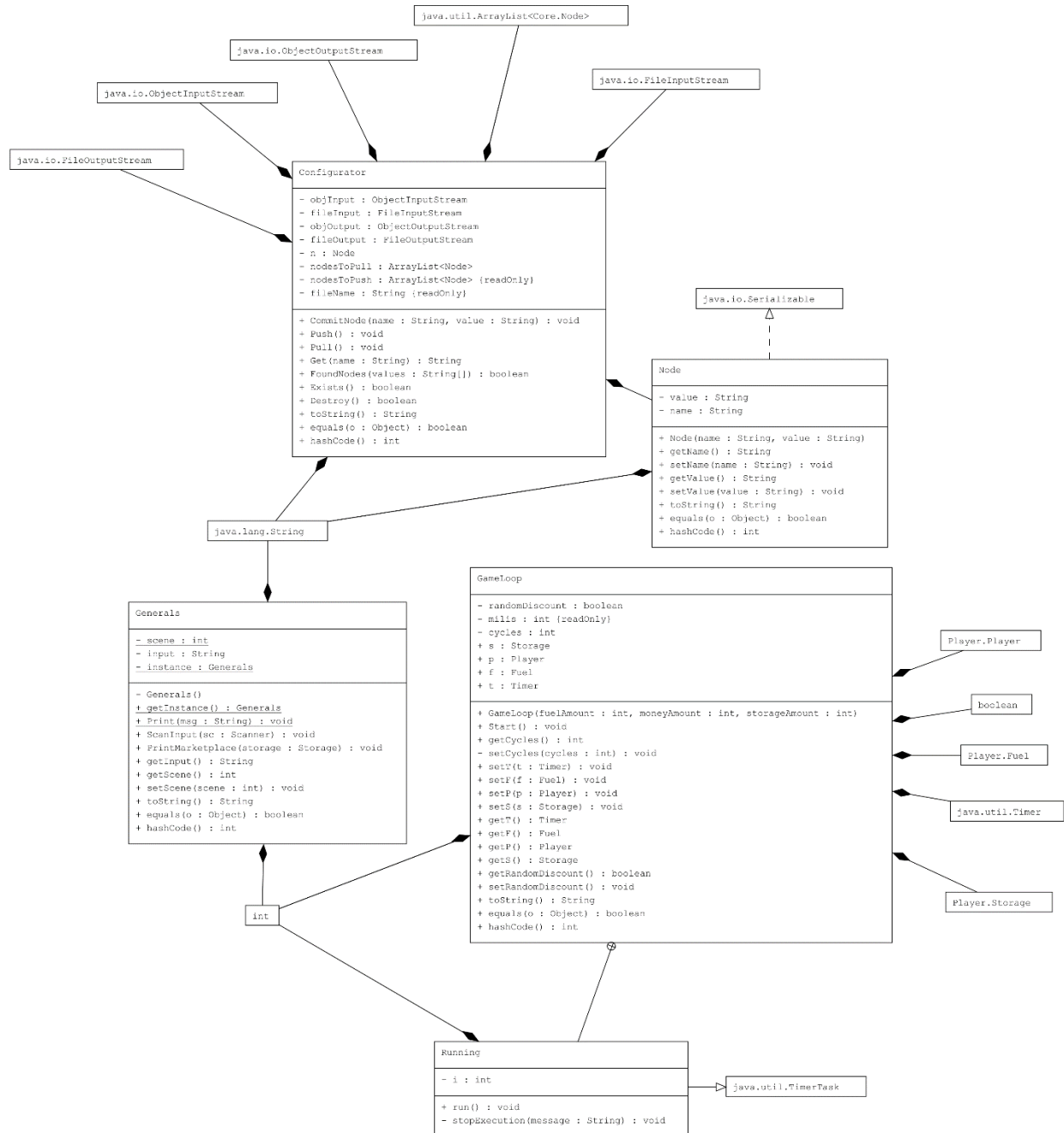
```

Využitie návrhového vzoru Singleton v triede Generals

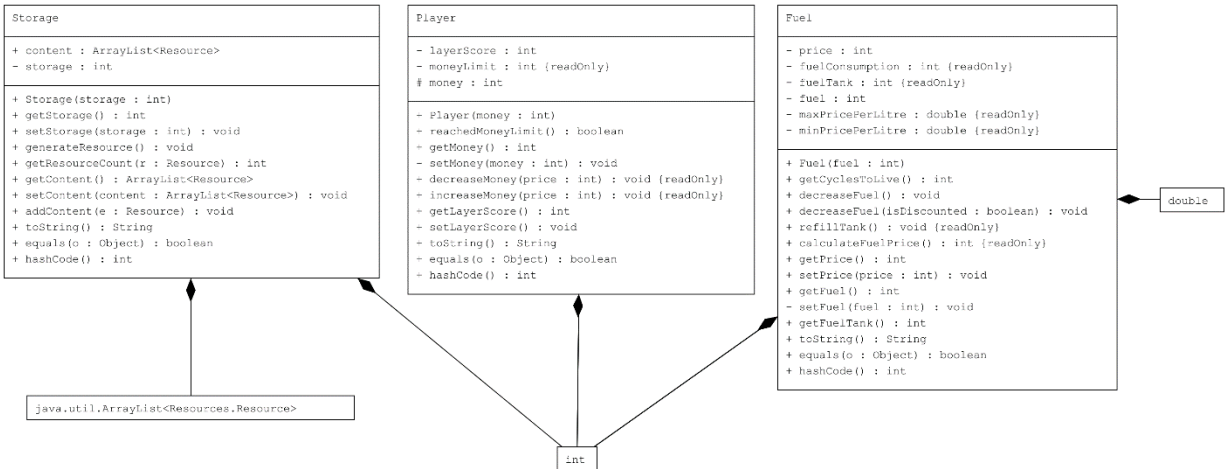
```
2 usages  👤 FreeDayZ
public class Main {
    👤 FreeDayZ
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Configurator config = new Configurator();
        GameLoop gameCycle;
        boolean isNew = config.FoundNodes(new String[]{"Money", "Storage", "Fuel"});
        //-----
        Generals g = Generals.getInstance();
        // -----
```

Ukážka získania inštancie pomocou návrhového vzoru Singleton v Main.java

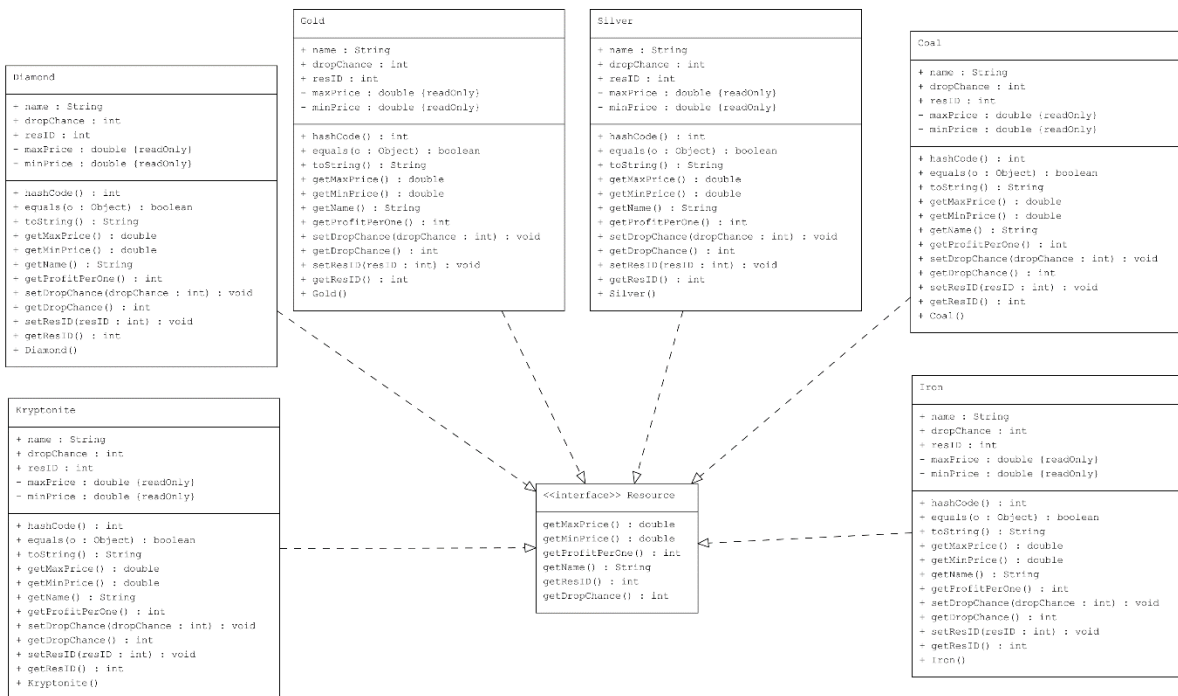
UML Diagramy tried programu



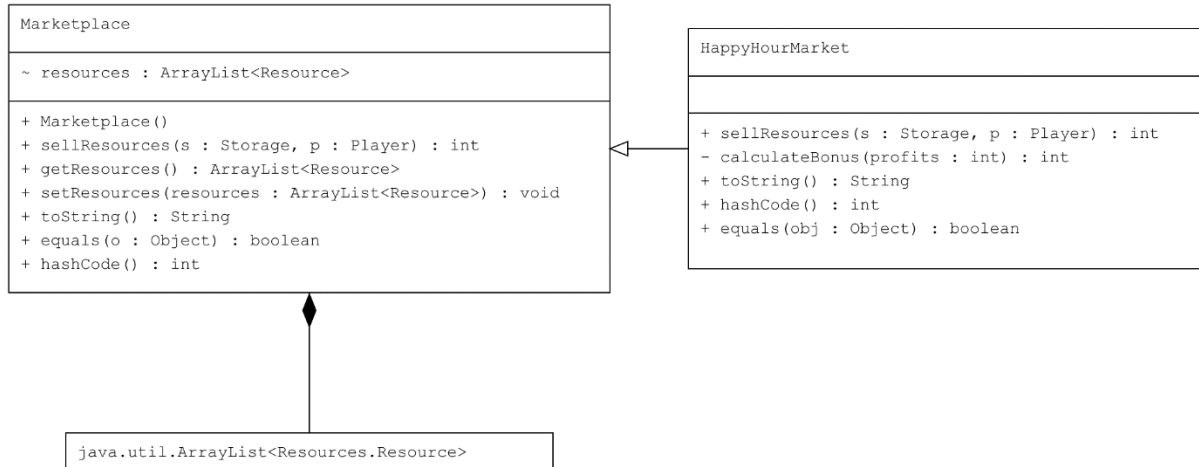
Package Core



Package Player



Package Resources



Package Scenes