

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

ZADANIE 3
POUŽITÉ OOP PRINCÍPY
FIIT HEROES

Základy objektovo-orientovaného programovania
Vyučujúci: doc. Ing. Ján Lang, PhD., Bc. Juraj Ďurej

Názov hry: FIIT HEROES

ZADANIE 2

- | | |
|---|---|
| 1. Implementovaná 3 kľúčové funkcionality podľa rámcového zadania. | 9 |
| 2. Aspoň dve zmysluplné hierarchie dedenia. | 1 |
| 3. Zapuzdrenie a správne modifikátory prístupu (členské premenné sú private). | 1 |
| 6. Použiť preťažovanie (overloading) metód | 1 |
| 7. Použiť prekonávanie (overriding) metód. | 1 |
| 8. Použiť vzťah agregácie. | 1 |
| 9. Použiť vzťah kompozície. | 1 |
| 10. Použiť vzťah asociácie. | 1 |
| 11. Použiť finálny atribút alebo triedu. | 1 |
| 12. Použiť finálnu metódu. | 1 |
| 13. Použiť abstraktnú triedu. | 1 |
| 14. Použiť abstraktnú metódu. | 1 |
| 15. Použiť statickú metódu. | 1 |
| 16. Použiť statický atribút. | 1 |
| 17. Implementácia vlastného rozhrania. | 1 |
| 18. Použitie upcasting-u. | 1 |
| 19. Použitie downcasting-u. | 1 |
| 20. Implementácia privátneho konštruktora alebo návrhového vzoru Singleton | 1 |
| 24. Vami navrhnutý UML diagram tried (všetky vzťahy a balíky, minimálne 6 tried pre získanie plného počtu bodov). | 8 |

ZADANIE 3

- | | |
|---|---|
| 4. Vhodná organizácia kódu do balíkov. | 1 |
| 5. Konzistentný štýl písania (pomenúvania) v zdrojovom kóde. | 1 |
| 21. Vyznačenie princípov v dokumente oopPouzite Principy.pdf (musia byť vyznačené). | 1 |
| 22. Korektne prekonané (overriden) metódy toString(), hashCode(), equals(Object obj) v každej Entite(entity sú triedy z ktorých vytvárame objekty, nieje to Service alebo Main class) | 1 |
| 23. Správne použitie názvov tried/metód + správnych konvencií(žiadne slovenské názvy alebo triedy napísané malým písmenom) | 1 |
| 25. Správne použitie dokumentácie kódu (javadoc + komentare kodu) | 1 |

OBSAH

1. Implementovaná 3 kľúčové funkcionality podľa rámcového zadania. (9)	4
2. Aspoň dve zmysluplné hierarchie dedenia. (1)	9
3. Zapuzdrenie a správne modifikátory prístupu (členské premenné sú private). (1)	11
4. Vhodná organizácia kódu do balíkov. (1)	11
5. Konzistentný štýl písania (pomenúvania) v zdrojovom kóde. (1).....	11
6. Použiť preťažovanie (overloading) metód (1)	12
7. Použiť prekonávanie (overriding) metód. (1).....	13
8. Použiť vzťah agregácie. (1)	13
9. Použiť vzťah kompozície. (1).....	14
10. Použiť vzťah asociácie. (1)	14
11. Použiť finálny atribút alebo triedu. (1)	15
12. Použiť finálnu metódu. (1).....	15
13. Použiť abstraktnú triedu. (1)	15
14. Použiť abstraktnú metódu. (1)	16
15. Použiť statickú metódu. (1)	16
16. Použiť statický atribút. (1).....	17
17. Implementácia vlastného rozhrania. (1).....	18
18. Použitie upcasting-u. (1).....	19
19. Použitie downcasting-u. (1).....	20
20. Implementácia privátneho konštruktora alebo návrhového vzoru Singleton (1)	21
21. Vyznačenie princípov v dokumente oopPouzite Principy.pdf (1)	22
22. Korektne prekonané (overriden) metódy toString(), hashCode(), equals(Object obj) v každej Entite(entity sú triedy z ktorých vytvárame objekty, nieje to Service alebo Main class) (1).....	22
23. Správne použitie názvov tried/metód + správnych konvencií(žiadne slovenské názvy alebo triedy napísané malým písmenom) (1).....	24
24. Vami navrhnutý UML diagram tried (všetky vzťahy a balíky, minimálne 6 tried pre získanie plného počtu bodov). (8)	25
25. Správne použitie dokumentácie kódu (javadoc + komentare kodu) (1)	26

1. Implementovaná 3 kľúčové funkcionality podľa rámcového zadania. (9)

(1.funkcionalita) – zvolenie a nastavenie obťažnosti

```
Funkcia setDifficulty() vypíše možnosti a nastaví náročnosť hry.
Možnosti náročnosti:
• ZAČIATOČNÍK
• POKROČILÝ

2 usages
50 public void setDifficulty(){
51     System.out.println("-----ZVOLTE NÁROČNOST HRY!-----");
52     System.out.println("      (A)-ZAČIATOČNÍK      (B)-POKROČILÝ");
53     System.out.println("-----");
54
55     String inputText = readTextFromConsole();
56     if(inputText.equals("A")){
57         System.out.println("Bola zvolená náročnosť \"ZAČIATOČNÍK\"");
58         System.out.println();
59         Main.difficulty = 2;
60     } else if (inputText.equals("B")) {
61         System.out.println("Bola zvolená náročnosť \"POKROČILÝ\"");
62         System.out.println();
63         Main.difficulty = 0;
64     }
65     else{
66         System.out.println("Nepodarilo sa zvoliť náročnosť, skúste to znovu! ");
67         System.out.println();
68         Main.difficulty = 333;
69         setDifficulty();
70     }
71
72 }
```

(2.funkcionalita) – načítanie mena hráča, vytvorenie hráča a nastavenie mena

```
Funkcia createPlayer() vypýta meno z konzoly a vytvorí hráča (Player).
Na začiatku hráč dostane zbraň a brnenie. Podľa zvolenej náročnosti hry sa upravuje sila zbrane.
See Also: Items
characters

1 usage
81 public void createPlayer(){
82     System.out.print("Zadajte meno hráča: ");
83
84     actualPlayer = new Player();
85     actualPlayer.setName(readTextFromConsole());
86
87     // AK JE NÁROČNOST "ZAČIATOČNÍK" tak sila zbrane + (DIFFICULTY_NUMBER * <5;10>)
88     if(Main.difficulty == 2) {
89         actualPlayer.weapon.setStrength(
90             actualPlayer.weapon.getStrength() + (DIFFICULTY_NUMBER * Person.random.nextInt( origin: 5, bound: 11))
91         );
92     }
```

(3.funkcionalita) – pri vytvorení hráča (Player) sa mu vytvorí zbraň a brnenie

```
Abstraktná trieda Person, z ktorej sa nedá vytvoriť inštancia, dá sa z nej iba dediť.  
13 usages 2 inheritors  
9 public abstract class Person {  
10  
11     protected String name; // prístupná v balíku characters,  
29 usages  
12     protected int health;  
11 usages  
13     public Weapon weapon = new Weapon();  
32 usages  
14     public Armor armor = new Armor();
```

v triede Person

```
1 usage  
40 public Armor(){  
41     String [] namesOfArmors = {  
42         "Zázračná prilba proti F(x)",  
43         "Magické nohavice nekonečných bodov",  
44         "Nepriestrelná vesta"  
45     };  
46     //Random random = new Random();  
47     //int randomInteger = random.nextInt(3); // GENERUJE NAHODNE 0, 1, 2  
48  
49     // VYBERIE NAHODNE NAZOV ZBRANE Z POLA namesOfWeapons  
50     //name = namesOfWeapons[random.nextInt(3)];  
51     setItemName(namesOfArmors[random.nextInt( bound: 3)]);  
52  
53     armor = (random.nextInt( origin: 10, bound: 31)); //CISLA <10;30>  
54 }
```

v triede Armor

```
1 usage  
36 public Weapon(){  
37     String [] namesOfWeapons = {  
38         "Analytická zbraň odmocnín",  
39         "Objektovo orientovaný meč",  
40         "Kniha čo reže vajcia"  
41     };  
42     //Random random = new Random();  
43     //int randomInteger = random.nextInt(3); // GENERUJE NAHODNE 0, 1, 2  
44  
45     // VYBERIE NAHODNE NAZOV ZBRANE Z POLA namesOfWeapons  
46     //name = namesOfWeapons[random.nextInt(3)];  
47     setItemName(namesOfWeapons[random.nextInt( bound: 3)]);  
48  
49     strength = (random.nextInt( origin: 70, bound: 91)); //CISLA <70;90>  
50 }
```

v triede Weapon

(4.funkcionalita) – v mainMenu pri zvolení možnosti „MOJA POSTAVA“ sa vypíšu informácie o hráčovi

```

Funkcia mainMenu() vypíše možnosti hry a umožňuje zvolenie možnosti.

• MOJA POSTAVA
• SUBOJ
• KONIEC

Voľba MOJA POSTAVA vypíše informácie o hráčovi ( Player )

Voľba SUBOJ vytvorí protivníka ( Enemy ), jeho život bude v náhodnom okolí hráča ( Player ).
Následne zavolá funkciu fight().

Voľba KONIEC ukončí hru a vypíše informácie o hre.

4 usages
112 public void mainMenu(){
113     System.out.println();
114     System.out.println("----- MENU -----");
115     System.out.println(" (A)-MOJA POSTAVA (B)-SUBOJ (X)-KONIEC");
116     System.out.println("-----");
117
118     String inputText = readTextFromConsole();
119
120     switch (inputText) {
121         case "A" -> {
122             // MOJA POSTAVA
123             if(actualPlayer instanceof Player){
124                 ((Player)actualPlayer).playerInfo();
125             }
126             mainMenu();
127         }
128         case "B" -> {
129             // SUBOJ

```

```

Funkcia playerInfo() vypíše informácie o hráčovi ( Player ).

1 usage
40 public void playerInfo(){
41     System.out.println("-----MOJA POSTAVA-----");
42     System.out.println("MENO: " + name);
43     System.out.println("POČET VÝHIER: " + numberOfWins);
44     System.out.println("MAXIMÁLNY ŽIVOT: " + health);
45     System.out.println("ZBRAŇ: \"\"+weapon.getItemName()+ \"\" (\"+weapon.getStrength()+\")");
46     System.out.println("BRNENIE: \"\"+armor.getItemName()+ \"\" (\"+armor.getArmor()+\")");
47     System.out.println("-----");
48 }

```

(5.funkcionalita) – vytvorenie Enemy (so životom v okolí hráča)

```

1 usage
23 public Enemy(Player actualPlayer){
24     name = createEnemyName();
25     // ŽIVOT HRÁČA + (10 * numberOfWins * <-2;3> )
26     this.health = ( actualPlayer.health ) + (10 * actualPlayer.numberOfWins * Person.random.nextInt( origin: -2, bound: 3) ) ;
27
28     System.out.println();
29     numberOfEnemy++;
30 }

```

(6.funkcionalita) – súboj (útok a obrana)

Funkcia attack() vykoná útok.
Returns: síla útoku (int)

2 usages

```
77 public int attack(){
78     return weapon.getStrength();
79 }
80
```

Funkcia defense(Person attacker,int damage) vykoná obranu.
Params: attacker – kto útočí (Person)
damage – síla poškodenia (int)
Náhodne sa vyberá z možností:

- vyhnutie sa zásahu (0)
- ubratie z panciera (1)
- ubratie zo života (2)

2 usages

```
93 public void defense(Person attacker,int damage){
94     int randomNumber = Person.random.nextInt( bound: 3);
95     //System.out.println("OBRANA: NAHODNE CISLO: "+randomNumber);
96
97     if(randomNumber == 0){
98         System.out.println(name + " sa uhol.");
99
100        System.out.print(name);
101        System.out.print(" (HP: "+health+"      ");
102        System.out.print(" (BRNENIE: "+armor.getArmor()+")");
103        System.out.println();
104
```

```
105        System.out.print(attacker.name);
106        System.out.print(" (HP: "+attacker.health+"      ");
107        System.out.print(" (BRNENIE: "+attacker.armor.getArmor()+")");
108        System.out.println();
109    }
110    else if (randomNumber == 1) {
111        System.out.println("PANCIER ZASIAHNUTÝ.");
112
113        if(armor.getArmor() > 0){           // AK EŠTE MÁŠ BRNENIE TAK UBERAJ Z NEHO
114            if( damage > armor.getArmor()){ // AK JE POSKODENIE VACŠIE AKO ZNESIE PANCIER
115                // UBERIEM CELY PANCIER, ZVYŠOK UBERIEM ZO ŽIVOTA
116                damage = damage - armor.getArmor();
117
118                System.out.print(name);
119                System.out.print(" (HP: "+health+"-"+damage+"      ");
120                System.out.print(" (BRNENIE: "+armor.getArmor()+"-"+armor.getArmor()+")");
121                System.out.println();
122
123                armor.setArmor(0);
124                health = health - damage;
125
126                System.out.print(attacker.name);
127                System.out.print(" (HP: "+attacker.health+"      ");
128                System.out.print(" (BRNENIE: "+attacker.armor.getArmor()+")");
129                System.out.println();
130            }

```

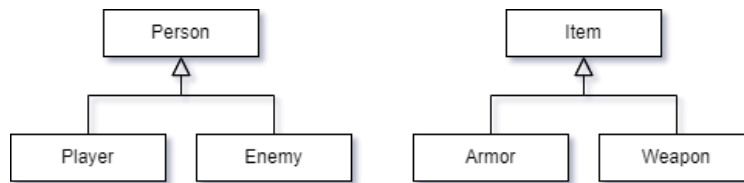
```

131     else if(damage == armor.getArmor()){ // AK JE POŠKODENIE ROVNAKÉ AKO BRNENIE
132         System.out.print(name);
133         System.out.print(" (HP: "+health+" )");
134         System.out.print(" (BRNENIE: "+armor.getArmor()+"-"+armor.getArmor()+") ");
135         System.out.println();
136
137         armor.setArmor(0);
138
139         System.out.print(attacker.name);
140         System.out.print(" (HP: "+attacker.health+" )");
141         System.out.print(" (BRNENIE: "+attacker.armor.getArmor()+") ");
142         System.out.println();
143     }
144     else{ // AK JE POŠKODENIE MENŠIE AKO PANCIER, IBA ODRÁTA Z PANCIERA
145         System.out.print(name);
146         System.out.print(" (HP: "+health+" )");
147         System.out.print(" (BRNENIE: "+armor.getArmor()+"-"+damage+" )");
148         System.out.println();
149
150         System.out.print(attacker.name);
151         System.out.print(" (HP: "+attacker.health+" )");
152         System.out.print(" (BRNENIE: "+attacker.armor.getArmor()+") ");
153         System.out.println();
154
155         armor.setArmor(armor.getArmor() - damage);
156     }
157     //this.armor.setBrnenie( this.armor.getArmor() - damage);
158 }
159 else{ // AK NEMAM BRNENIE TAK UBERAJ PRIAMO ŽIVOT
160     System.out.println("ZÁSAH ! brnenie más rozbité, uberám zo života.");
161     System.out.print(name);
162     System.out.print(" (HP: "+health+"-"+damage+" )");
163     System.out.print(" (BRNENIE: "+armor.getArmor()+") ");
164     System.out.println();
165
166     System.out.print(attacker.name);
167     System.out.print(" (HP: "+attacker.health+" )");
168     System.out.print(" (BRNENIE: "+attacker.armor.getArmor()+") ");
169     System.out.println();
170
171     health = health - damage;
172 }
173 }
174 else { // UBRATIE ZO ŽIVOTA
175     System.out.println("ZÁSAH ! uberám zo života.");
176     System.out.print(name);
177     System.out.print(" (HP: "+health+"-"+damage+" )");
178     System.out.print(" (BRNENIE: "+armor.getArmor()+") ");
179     System.out.println();
180
181     System.out.print(attacker.name);
182     System.out.print(" (HP: "+attacker.health+" )");
183     System.out.print(" (BRNENIE: "+attacker.armor.getArmor()+") ");
184     System.out.println();
185
186     health = health - damage;
187 }

```


2. Aspoň dve zmysluplné hierarchie dedenia. (1)

Triedy Player a Enemy dedia od Person. Triedy Armor a Weapon dedia od Item.



Trieda Item je zároveň abstraktná, nedá sa z nej vytvoriť inštancia, dá sa iba dediť.

```
Abstraktná trieda Item, z ktorej sa nedá vytvoriť inštancia, dá sa z nej iba dediť.
2 usages 2 inheritors
8 public abstract class Item {
    2 usages
    9 private String name;
    4 usages
    10 static final Random random = new Random();

    Funkcia getItemName() vypýta meno Itemu.
    Returns: name - názov Itemu (String)

    8 usages
    15 public final String getItemName(){
    16     return this.name;
    17 }
    18

    Funkcia setItemName(String name) nastaví názov Itemu.
    Params: name - názov Itemu (String)

    2 usages
    23 public final void setItemName(String name){
    24     this.name = name;
    25 }
```

```
Trieda Armor dedí od triedy Item.
Brnenie "chráni" hráča Playera alebo protivníka Enemy.
6 usages
9 public class Armor extends Item{
    6 usages
    10 private int armor;
    11

    Funkcia getArmor() vráti silu panciera.
    Returns: armor - sila panciera (int)

    23 usages
    16 public int getArmor() { return armor; }
    19

    Funkcia setArmor(int armor) nastaví silu brnenia.
    3 usages
    23 public void setArmor(int armor) {
    24     this.armor = armor;
    25

    26     if(armor == 0){
    27         System.out.println("BRNENIE BOLO ZNIČENÉ !");
    28     }
    29 }
```

```
Trieda Weapon dedí od triedy Item.
Pomocou zbrane môže Person útočiť.
6 usages
8 public class Weapon extends Item {
    9
    6 usages
    10 private int strength;
    11

    Funkcia getStrength() vráti silu zbrane.
    Returns: strength - sila zbrane (int)

    6 usages
    16 public int getStrength() { return strength; }
    19

    Funkcia setStrength(int sila) nastaví silu zbrane.
    1 usage
    23 public void setStrength(int sila) { this.strength = sila; }
```

Rovnako trieda Person je abstraktná.

```
Abstraktná trieda Person, z ktorej sa nedá vytvoriť inštancia, dá sa z nej iba dediť.
13 usages 2 inheritors
9 public abstract class Person {
10
11     protected String name; // prístupný v balíku characters, aj v triedach ktoré dedia z Person
12     protected int health;
13     public Weapon weapon = new Weapon();
14     public Armor armor = new Armor();
15     public static final Random random = new Random(); // static - patrí celej PERSON; final - neupravuje sa ďalej
16 }
```

Triedy player a Enemy dedia od triedy Person

```
Trieda Player dedí od triedy Person.
Je to aktuálny hráč, ktorý hraje hru.
7 public class Player extends Person {
8
9     7 usages
10     public int numberOfWins = 0;
11
12     // KONŠTRUKTOR
13     1 usage
14     public Player(){
15         health = 100;
16     }
17
18     // KONŠTRUKTOR
19     public Player(String name){
20         this.name = name;
21         health = 100;
22         System.out.println("Hráč "+name+" bol vytvorený!");
23     }
24 }
```

```
Trieda Enemy dedí od triedy Person.
Je to protivník, ktorý bude bojovať proti hráčovi Player.
7 public class Enemy extends Person {
8
9     5 usages
10     public static int numberOfEnemy = 0;
11
12     // KONŠTRUKTOR
13     public Enemy(){...}
14
15     // KONŠTRUKTOR
16     public Enemy(int health){...}
17
18     // KONŠTRUKTOR
19     public Enemy(Player actualPlayer){...}
20 }
```

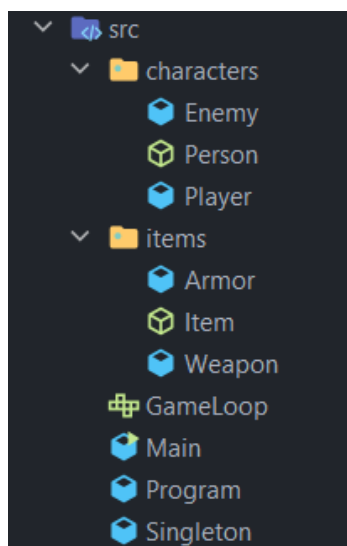
3. Zapuzdrenie a správne modifikátory prístupu (členské premenné sú private). (1)

Napríklad v triede Item mám atribút private String name (je dostupný iba v tejto triede) ak by som ho chcel použiť v triede ktorá dedí z ITEM, musím použiť (GETTER, SETTER METÓDY) v tomto prípade getItemName() a setItemName() ktoré sú už public.

```
Abstraktná trieda Item, z ktorej sa nedá vytvoriť inštancia, dá sa z nej iba dediť.  
2 usages 2 inheritors  
8 public abstract class Item {  
    2 usages  
    9 private String name;  
    4 usages  
    10 static final Random random = new Random();  
    Funkcia getItemName() vypýta meno Itemu.  
    Returns: name - názov Itemu (String)  
    8 usages  
    15 public final String getItemName(){  
    16     return this.name;  
    17 }  
    18  
    Funkcia setItemName(String name) nastaví názov Itemu.  
    Params: name - názov Itemu (String)  
    2 usages  
    23 public final void setItemName(String name){  
    24     this.name = name;  
    25 }
```

4. Vhodná organizácia kódu do balíkov. (1)

Kód som štrukturoval do balíkov nasledovne.



5. Konzistentný štýl písania (pomenúvania) v zdrojovom kóde. (1)

Dodržiaval som zásady správneho písania kódu.

názvy premenných = malými písmenami (camelCase)

názov triedy = prvé veľké písmeno (ProgramLogic, Main...)

metódy = malými písmenami (getName(), winning()...)

názvy, metódy, premenné v anglickom jazyku

6. Použiť preťažovanie (overloading) metód (1)

```
Enemy fighter = null;
//if(actualPlayer.numberOfWins == 0){
if( ((Player)actualPlayer).numberOfWins == 0 ){
    fighter = new Enemy( health: 50);
}
else{
    //fighter = new Enemy(actualPlayer);
    fighter = new Enemy( ((Player)actualPlayer) );
}
```

V trede Program pri prvej bitke vytvorím Enemy so životom 50 (Enemy fighter = new Enemy(50);)
Pri ďalších bitkách už vytváram Enemy, ktorý má život o veľkosti náhodného okolia hráča (posielam už hráča)

```
8 usages
5 public class Enemy extends Person {
6
7     5 usages
8     public static int numberOfEnemy = 0;
9
10    // KONŠTRUKTOR
11    public Enemy(){
12        health = 100;
13        numberOfEnemy++; // POČITADLO VSETKYCH ENEMY
14    }
15
16    1 usage
17    public Enemy(int health){
18        name = createEnemyName();
19        this.health = health;
20
21        numberOfEnemy++; // POČITADLO VSETKYCH ENEMY
22    }
23
24    1 usage
25    public Enemy(Player actualPlayer){
26        name = createEnemyName();
27        // ŽIVOT HRÁČA + (10 * numberOfWins * <-2;3> )
28        this.health = ( actualPlayer.health ) + (10 * actualPlayer.numberOfWins * Person.random.nextInt( origin: -2, bound: 3) ) ;
29
30        System.out.println();
31        numberOfEnemy++; // POČITADLO VSETKYCH ENEMY
32    }
33 }
```

PREŤAŽOVANIE KONŠTRUKTORA, konštruktor je špeciálny typ metódy

Enemy fighter = new Enemy(50); sem posielam int

Enemy fighter = new Enemy(actualPlayer); sem posielam Player

7. Použiť prekonávanie (overriding) metód. (1)

V triede Player prekonávam metódu setName(), pretože od nej vyžadujem iné správanie ako má Person (z ktorého Player dedí)

V triede Person:

```
Funkcia setName(String name) nastaví meno.  
Params: name – meno osoby ( Person )  
  
1 usage 1 override  
public void setName(String name) {  
    this.name = name;  
}
```

V triede Player:

```
Funkcia setName(String name) nastaví meno a realizuje kontrolný výpis.  
Params: name – meno osoby ( Person )  
  
1 usage  
@Override  
public void setName(String name) {  
    this.name = name;  
    System.out.println("Meno "+name+" bolo pridelené hráčovi!");  
}
```

8. Použiť vzťah agregácie. (1)

V triede Person vytváram triedy Weapon a Armor (Person has Weapon, Armor)

Enemy je typu person ak zomrie a zbraň alebo brnenie je horšie, tak Player si ju vezme
Weapon alebo Armor môžu žiť ďalej.

```
Abstraktná trieda Person, z ktorej sa nedá vytvoriť inštancia, dá sa z nej iba dediť.  
  
13 usages 2 inheritors  
9 public abstract class Person {  
10  
11     protected String name; // prístupný v balíku characters, aj v triedach ktoré dedia z Person  
29 usages  
12     protected int health;  
11 usages  
13     public Weapon weapon = new Weapon();  
32 usages  
14     public Armor armor = new Armor();  
}
```

V class Player na riadku 60, 65 pri splnení podmienok Weapon/Armor žijú aj po zabití Enemy

```
Funkcia updateStats(Enemy enemy) zavola funkciu winning() a skontroluje či je zbraň alebo brnenie  
lepšie, ak áno tak ich priradí hráčovi.  
Params: enemy – protivník, ktorý bol porazený v súboji  
  
1 usage  
54 @Override  
55 public void updateStats(Enemy enemy){  
56     winning();  
57  
58     // AK JE ZBRAN PROTIVNIKA LEPSIA, TAK VYMEM  
59     if( enemy.weapon.getStrength() > (this.weapon.getStrength()) ){  
60         System.out.println("ZBRAN PROTIVNIKA JE LEPSIA, vezmem si ju...");  
61         this.weapon = enemy.weapon.getWeapon();  
62     }  
63     // AK JE BRNENIE PROTIVNIKA LEPSIE, TAK VYMEM  
64     if( enemy.armor.getArmor() > this.armor.getArmor() ){  
65         System.out.println("BRNENIE PROTIVNIKA JE LEPSIE, vezmem si ho...");  
66         this.armor = enemy.armor.getArmorItem();  
67     }  
}
```

9. Použití vzťah kompozície. (1)

Trieda Program bez Player-a nemá zmysel.

Keď player zomrie, končí sa aj program, nevie existovať bez playera.

```
Program implementuje INTERFACE GameLoop.  
Je to hlavný cyklus hry.  
Hra pozostáva z Hráča ( PLayer ) ktorý bojuje proti protivníkom ( Enemy )  
  
2 usages  
9 public class Program implements GameLoop {  
    16 usages  
10     Person actualPlayer = null;
```

```
Funkcia createPlayer() vypýta meno z konzoly a vytvorí hráča ( Player ).  
Na začiatku hráč dostane zbraň a brnenie. Podľa zvolenej náročnosti hry sa upravuje sila zbrane.  
See Also: items,  
          characters  
  
1 usage  
81 public void createPlayer(){  
82     System.out.print("Zadajte meno hráča: ");  
83  
84     actualPlayer = new Player();  
85     actualPlayer.setName(readTextFromConsole());  
86  
87     // AK JE NÁROČNOST "ZAČIATOČNÍK" tak sila zbrane + (DIFFICULTY_NUMBER * <5;10>)  
88     if(Main.difficulty == 2) {  
89         actualPlayer.weapon.setStrength(  
90             actualPlayer.weapon.getStrength() + (DIFFICULTY_NUMBER * Person.random.nextInt( origin: 5, bound: 11))  
91         );  
92     }
```

10. Použití vzťah asociácie. (1)

AGREGÁCIA je zároveň ASOCIÁCIA alebo class Main má class Program

```
FIIT HEROES  
Author: Adam Vrabel  
Version: ZADANIE 3  
  
6 usages  
public class Main {  
    6 usages  
    public static int difficulty = 333;  
  
    public static void main(String[] args) {  
        System.out.println("ADAM VRABEL, ZADANIE ZOOP");  
        System.out.println("FIIT HEROES");  
        System.out.println();  
  
        Program run = new Program();  
  
        run.setDifficulty();  
        run.createPlayer();  
        run.mainMenu();  
  
        ///run.trySingleton();  
        ///run.justTry();  
    }  
}
```

11. Použiť finálny atribút alebo triedu. (1)

V triede Person - `public static final Random random = new Random();` Nebudem meniť jeho metódy, iba používať

V triede Program používam DEFFICULTY_NUMBER ako konštantu pri výpočtoch pre zvolenú obťažnosť „POKROČILÝ“.

```
1 usage
private static final int DIFFICULTY_NUMBER = 15; // KONŠTANTA (final)
```

```
62 // AK JE NAROCNOST "ZAČIATOČNÍK" tak sila zbrane + (DIFFICULTY_NUMBER * <5;10>)
63 if(Main.difficulty == 2) {
64     actualPlayer.weapon.setStrength(
65         actualPlayer.weapon.getStrength() + (DIFFICULTY_NUMBER * Person.random.nextInt( origin: 5, bound: 11))
66     );
```

12. Použiť finálnu metódu. (1)

Finálne metódy v triede Item som použil preto, lebo už nechcem dovoliť aby sa metódy `getItemName()` a `setItemName()` prekonávali (OVERRIDE)

```
Abstraktná trieda Item, z ktorej sa nedá vytvoriť inštancia, dá sa z nej iba dediť.
2 usages 2 inheritors
8 public abstract class Item {
    2 usages
    9     private String name;
    4 usages
    10     static final Random random = new Random(); // STATIC - PATRÍ CELEJ TRIEDE, nie inštanciám (final, nebudem ju meniť
    11
    Funkcia getItemName() vypýta meno Itemu.
    Returns: name - názov Itemu (String)
    8 usages
    16     public final String getItemName(){
    17         return this.name;
    18     }
    19
    Funkcia setItemName(String name) nastaví názov Itemu.
    Params: name - názov Itemu (String)
    2 usages
    24     public final void setItemName(String name){
    25         this.name = name;
    26     }
    27
    28 }
```

13. Použiť abstraktnú triedu. (1)

PERSON A ITEM nedá sa z nich vytvoriť inštancia, sme sa z nich iba dediť

```
public abstract class Person
```

```
public abstract class Item
```

14. Použiť abstraktnú metódu. (1)

V abstraktnej triede Person sa nachádza abstraktná metóda winning(), každý kto dedí z triedy Person musí mať napísanú implementáciu metódy winning() (riadok 53)

```
Abstraktná funkcia winning().  
Každý kto dedí z triedy Person musí mať napísanú implementáciu funkcie winning().  
  
2 usages 2 implementations  
53 public abstract void winning(); // KAZDY kto dedí z PERSON musí mať implementáciu winning()
```

15. Použiť statickú metódu. (1)

V triede Person je to metóda **public static void introduceYourself(Person person)**

Táto metóda patrí triede Person, nie jej inštanciam (respektíve patrí potomkom ktorý z nej dedia ale nie jeho inštanciam)

```
Statická funkcia introduceYourself(Person person) vypíše informácie o hráčovi Player a protivníkovi hráča Enemy.  
Params: person – aktuálny hráč  
  
2 usages  
63 public static void introduceYourself(Person person){  
64     if(person instanceof Player){  
65         System.out.print("#HRÁČ ");  
66     }  
67     else if(person instanceof Enemy){  
68         System.out.print("#PROTIVNÍK ");  
69     }  
70     System.out.println(person.name+" HP("+person.health+") ZBRAŇ: \""+person.weapon.getItemName()+ "\"("+person.weapon.ge  
71 }
```

POUŽITIE V KÓDE

```
Person.introduceYourself(actualPlayer);  
Person.introduceYourself(fighter);
```


16. Použiť statický atribút. (1)

Trieda Main má statický atribút difficulty, ktorý patrí triede Main

```
FIIT HEROES
Author: Adam Vrabel
Version: ZADANIE 3

6 usages
public class Main {
    6 usages
    public static int difficulty = 333;

    public static void main(String[] args) {
        System.out.println("ADAM VRABEL, ZADANIE ZOOP");
        System.out.println("FIIT HEROES");
        System.out.println();

        Program run = new Program();

        run.setDifficulty();
        run.createPlayer();
        run.mainMenu();

        //run.trySingleton();
        //run.justTry();
    }
}
```

alebo

trieda Person má static final Random random (ktorý je aj statický aj finálny zároveň) (RIADOK 12)

```
Abstraktná trieda Person, z ktorej sa nedá vytvoriť inštancia, dá sa z nej iba dediť.
13 usages 2 inheritors
9 public abstract class Person {
10
11     protected String name; // prístupný v balíku characters, aj v triedach ktoré dedia z Person
    29 usages
12     protected int health;
    11 usages
13     public Weapon weapon = new Weapon();
    32 usages
14     public Armor armor = new Armor();
    4 usages
15     public static final Random random = new Random(); // static - patrí celej PERSON; final-neupravuje sa ďalej
16 }
```

17. Implementácia vlastného rozhrania. (1)

Rozhranie som použil ako šablónu na vytvorenie cyklu hry.

```
1 usage 1 implementation
4 ✖ public interface GameLoop {
5     Player actualPlayer = null;
6
7     3 usages 1 implementation
8     String readTextFromConsole();
9     2 usages 1 implementation
10    void setDifficulty();
11    1 usage 1 implementation
12    void createPlayer();
13    4 usages 1 implementation
14    void mainMenu();
15 }
```

Následne v triede Program implementujem rozhranie GameLoop, táto trieda program musí mať všetky implementácie ktoré má rozhranie GameLoop.

```
2 usages
public class Program implements GameLoop {
```

```
Program implementuje INTERFACE GameLoop.
Je to hlavný cyklus hry.
Hra pozostáva z Hráča ( Player ) ktorý bojuje proti protivníkovi ( Enemy )

2 usages
9 public class Program implements GameLoop {
10     16 usages
11     Person actualPlayer = null;
12     1 usage
13     private static final int DIFFICULTY_NUMBER = 15; // KONŠTANTA (final)
14     3 usages
15     Singleton programDatabase = Singleton.getInstance(); //GET INSTANCE SINGLETON DATABASE
16
17     public void trySingleton(){...}
18     public void justTry(){...}
19
20     Funkcia readTextFromConsole() načíta text z konzole.
21     Returns: vráti načítaný text (typu String)
22
23     3 usages
24     public String readTextFromConsole(){...}
25
26     Funkcia setDifficulty() vypíše možnosti a nastaví náročnosť hry.
27     Možnosti náročnosti:
28     • ZAČIATOČNÍK
29     • POKROČILÝ
30
31     2 usages
32     public void setDifficulty(){...}
```

```

81  fx  public void createPlayer(){...}
98
    Funkcia mainMenu() vypíše možnosti hry a umožňuje zvolenie možnosti.
    • MOJA POSTAVA
    • SUBOJ
    • KONIEC

    Volba MOJA POSTAVA vypíše informácie o hráčovi ( Player )
    Volba SUBOJ vytvorí protivníka ( Enemy ), jeho život bude v náhodnom okolí hráča ( Player ).
    Následne zavolá funkciu fight().

    Volba KONIEC ukončí hru a vypíše informácie o hre.

    4 usages
112  fx  public void mainMenu(){...}
187
    Funkcia fight() nasimuluje súboj medzi hráčom ( Player ) a protivníkom ( Enemy ).
    Súboj končí pokiaľ jeden z bojovníkov neminie svoj život.
    Params:  player – hráč ktorý bojuje
             enemy – protivník, ktorý sa bráni a následne útočí hráča
    See Also: Person

    1 usage
195  @  private void fight(Player player, Enemy enemy){...}
222
223  }

```

18. Použitie upcasting-u. (1)

Na začiatku v triede Program vytvorím objekt typu Person actualPlayer.

Následne v triede Program v metóde createPlayer(), pri načítaní mena podľa obťažnosti zapisujem (vytváram cez konštruktor) do actualPlayer čo je typu Person (PARENT) jeho potomka Player (CHILD)

```

4  2 usages
    public class Program implements GameLoop {
        16 usages
5      Person actualPlayer = null;

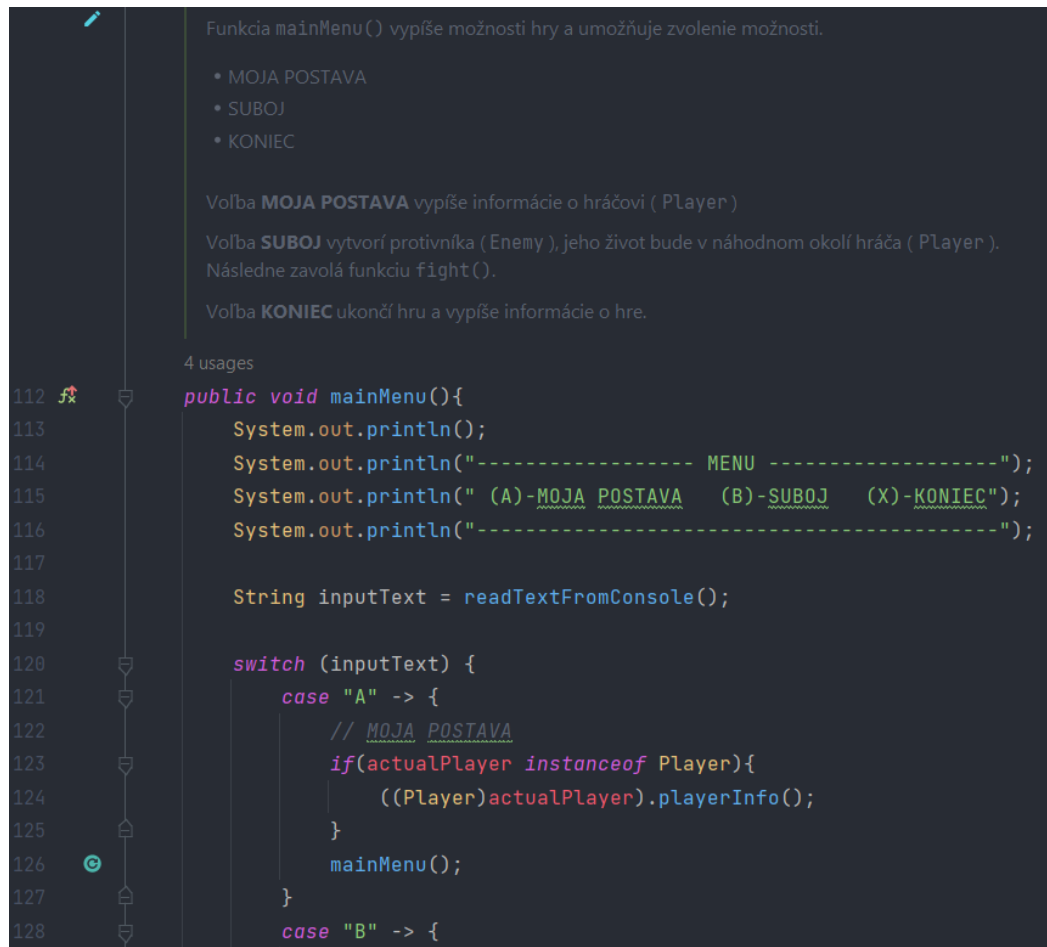
    Funkcia createPlayer() vypýta meno z konzoly a vytvorí hráča ( Player ).
    Na začiatku hráč dostane zbraň a brnenie. Podľa zvolenej náročnosti hry sa upravuje sila zbrane.
    See Also: Items, characters

    1 usage
81  fx  public void createPlayer(){
82      System.out.print("Zadajte meno hráča: ");
83
84      actualPlayer = new Player();
85      actualPlayer.setName(readTextFromConsole());
86
87      // AK JE NÁROČNOSŤ "ZAČIATOČNÍK" tak sila zbrane = (DIFFICULTY_NUMBER * <5;10>)
88      if(Main.difficulty == 2) {
89          actualPlayer.weapon.setStrength(
90              actualPlayer.weapon.getStrength() + (DIFFICULTY_NUMBER * Person.random.nextInt( origin: 5, bound: 11))
91          );
92      }
93      //System.out.println(actualPlayer.weapon.name);
94      //System.out.println(actualPlayer.weapon.getItemName());
95      //System.out.println(actualPlayer.weapon.getStrength());
96      //System.out.println(actualPlayer.name);
97  }

```

19. Použitie downcasting-u. (1)

Downcasting používam, aby som mohol použiť metódu ktorú má iba Player, keďže actualPlayer je typu Person. Pomocou podmienky zabezpečím že výpis „playerInfo“ prebehne iba v prípade že actualPlayer bude inštanciou Player. (riadok 123)



```
Funkcia mainMenu() vypíše možnosti hry a umožňuje zvolenie možnosti.

• MOJA POSTAVA
• SUBOJ
• KONIEC

Volba MOJA POSTAVA vypíše informácie o hráčovi ( Player )
Volba SUBOJ vytvorí protivníka ( Enemy ), jeho život bude v náhodnom okolí hráča ( Player ).
Následne zavolá funkciu fight().
Volba KONIEC ukončí hru a vypíše informácie o hre.

4 usages
112 public void mainMenu(){
113     System.out.println();
114     System.out.println("----- MENU -----");
115     System.out.println(" (A)-MOJA POSTAVA (B)-SUBOJ (X)-KONIEC");
116     System.out.println("-----");
117
118     String inputText = readTextFromConsole();
119
120     switch (inputText) {
121         case "A" -> {
122             // MOJA POSTAVA
123             if(actualPlayer instanceof Player){
124                 ((Player)actualPlayer).playerInfo();
125             }
126             mainMenu();
127         }
128         case "B" -> {
```

20. Implementácia privátneho konštruktora alebo návrhového vzoru Singleton (1)

Návrhový vzor **Singleton** zabezpečí že v programe bude vždy iba jedna inštancia.
Ak ešte inštancia nebola vytvorená tak ju vytvorí.
Ak už inštancia bola vytvorená tak ju iba vráti.

12 usages

```
9 public class Singleton {
10     3 usages
11     private static Singleton instance = null;
12     1 usage
13     private Singleton(){}
14     3 usages
15     public static Singleton getInstance(){
16         if(instance == null){
17             instance = new Singleton();
18         }
19         return instance;
20     }
21 }
22 6 usages
23 private LinkedList<String> namesOfEnemy = new LinkedList<>();
24
25 Funkcia addEnemy(Object thing) skontroluje či je parameter Enemy alebo String a priradí do databázy
26 meno bojovníka.
27 Params: thing – typu Object
28
29 1 usage
30 public void addEnemy(Object thing){
31     if(thing instanceof Enemy){
32         namesOfEnemy.push(((Enemy)thing).getName()); // PUSH NAME OF ENEMY TO DATABASE
33     }
34     else if (thing instanceof String) {
35         namesOfEnemy.push((String)thing); // PUSH STRING TO DATABASE
36     }
37 }
```

Program implementuje INTERFACE GameLoop.
Je to hlavný cyklus hry.
Hra pozostáva z Hráča (Player) ktorý bojuje proti protivníkovi (Enemy)

2 usages

```
9 public class Program implements GameLoop {
10     16 usages
11     Person actualPlayer = null;
12     1 usage
13     private static final int DIFFICULTY_NUMBER = 15; // KONŠTANTA (final)
14     3 usages
15     Singleton programDatabase = Singleton.getInstance(); //GET INSTANCE SINGLETON DATABASE
```

V triede Program na riadku 12 si vypýtam inštanciu mojej Singletone databázy, návrhový vzor Singleton zabezpečí to, ak ešte jediná inštancia nie je vytvorená, tak ju vytvorí v opačnom prípade iba vráti inštanciu už vytvorenú.

Následne v triede Program na riadku 218 vo funkcii fight pri porazení Protivníka zapíšem meno porazeného do databázy

```
215 if(player.getHealth() > enemy.getHealth()){ // AK VYHRAL HRAC
216     player.updateStats(enemy);
217
218     programDatabase.addEnemy(enemy); // PRIDANIE MENA PORAZENÉHO ENEMY DO ZOZNAMU
219     //databaseInsideFight.doPrint();
220 }
```

Pri zabití Player-a alebo ukončení hry volám výpis porazených protivníkov na riadku 165 a 178.

```
159     else{ // AK PLAYER UZ ZOMREL, VYHRAL FIGHTER
160         System.out.println("-----GAME - INFO-----");
161         fighter.winning();
162         //actualPlayer.playerInfo();
163         System.out.println("Meno hráča: "+actualPlayer.getName());
164         System.out.println("Bojoval si proti "+Enemy.numberOfEnemy+" protivníkom");
165         programDatabase.doPrint(); // VYPIS ZO SINGLETONU
166         //System.out.println("Počet vyhratých bitiek: "+actualPlayer.numberOfWeeks);
167         System.out.println("Počet vyhratých bitiek: "+((Player)actualPlayer).numberOfWins);
168         System.out.println("-----");
169     }
170
171 }
172
173 case "X" -> {
174     System.out.println("HRA SKONČILA !");
175     System.out.println("-----GAME - INFO-----");
176     System.out.println("Meno hráča: "+actualPlayer.getName());
177     System.out.println("Bojoval si proti "+Enemy.numberOfEnemy+" protivníkom");
178     programDatabase.doPrint(); // VYPIS ZO SINGLETONU
179     //System.out.println("Počet vyhratých bitiek: "+actualPlayer.numberOfWeeks);
180     System.out.println("Počet vyhratých bitiek: "+((Player)actualPlayer).numberOfWins);
181     System.out.println("-----");
182 }
```

21. Vyznačenie princípov v dokumente oopPouzite Principy.pdf (1)

Princípy sú vyznačené a opísané v tomto dokumente.

22. Korektne prekonané (overriden) metódy toString(), hashCode(), equals(Object obj) v každej Entite(entity sú triedy z ktorých vytvárame objekty, nieje to Service alebo Main class) (1)

V triede Singleton

```
50     @Override
51     public int hashCode() {
52         return ("FIIT STU").hashCode() + super.hashCode();
53     }
54     @Override
55     public boolean equals(Object obj) {
56         if(obj instanceof Singleton){
57             if(obj.hashCode() == this.hashCode()){
58                 return true;
59             }
60             else {
61                 return false;
62             }
63         }
64         else{
65             return false;
66         }
67     }
68     @Override
69     public String toString() {
70         return "SINGLETON_"+ super.toString();
71     }
```

V triede Player

```
80      @Override
81      public int hashCode() {
82          return name.hashCode() * weapon.hashCode() * armor.hashCode() + numberOfWins;
83      }
84
85      @Override
86      public boolean equals(Object obj) {
87          if(obj instanceof Player){
88              if ( ( ((Person)obj).name.hashCode() == this.name.hashCode()) && (((Person)obj).weapon.hashCode() == this.weapon.
89                  return true;
90              }
91              else{
92                  return false;
93              }
94          }
95          else{
96              return false;
97          }
98      }
99      @Override
100     public String toString() {
101         return "PLAYER_"+ super.toString();
102     }
103 }
```

V triede Enemy

```
54      @Override
55      public int hashCode() {
56          return name.hashCode() * weapon.hashCode() * armor.hashCode() + numberOfEnemy;
57      }
58
59      @Override
60      public boolean equals(Object obj) {
61          if(obj instanceof Enemy){
62              if ( ( ((Enemy)obj).name.hashCode() == this.name.hashCode()) && (((Enemy)obj).weapon.hashCode() == this.weapon.
63                  return true;
64              }
65              else{
66                  return false;
67              }
68          }
69          else{
70              return false;
71          }
72      }
73      @Override
74      public String toString() {
75          return "ENEMY_"+ super.toString();
76      }
77  }
```

V triede Armor

```
57      @Override
58      public int hashCode() {
59          return armor * getItemName().hashCode();
60      }
61
62      @Override
63      public boolean equals(Object obj) {
64          if(obj instanceof Armor){
65              if ( ( ((Armor)obj).armor == this.armor) && (obj.hashCode() == this.hashCode()) ){
66                  return true;
67              }
68              else{
69                  return false;
70              }
71          }
72          else{
73              return false;
74          }
75      }
76      @Override
77      public String toString() {
78          return "ARMOR_"+ super.toString();
79      }
80  }
```

V triede Weapon

```
52      @Override
53      public int hashCode() {
54          return strength * getItemName().hashCode();
55      }
56
57      @Override
58      public boolean equals(Object obj) {
59          if(obj instanceof Weapon){
60              if ( ( (Weapon)obj).strength == this.strength) && (obj.hashCode() == this.hashCode()) ){
61                  return true;
62              }
63              else{
64                  return false;
65              }
66          }
67          else{
68              return false;
69          }
70      }
71      @Override
72      public String toString() {
73          return "WEAPON_" + super.toString();
74      }
75  }
```

23. Správne použitie názvov tried/metód + správnych konvencií(žiadne slovenské názvy alebo triedy napísané malým písmenom) (1)

Dodržoval som zásady správneho písania kódu.

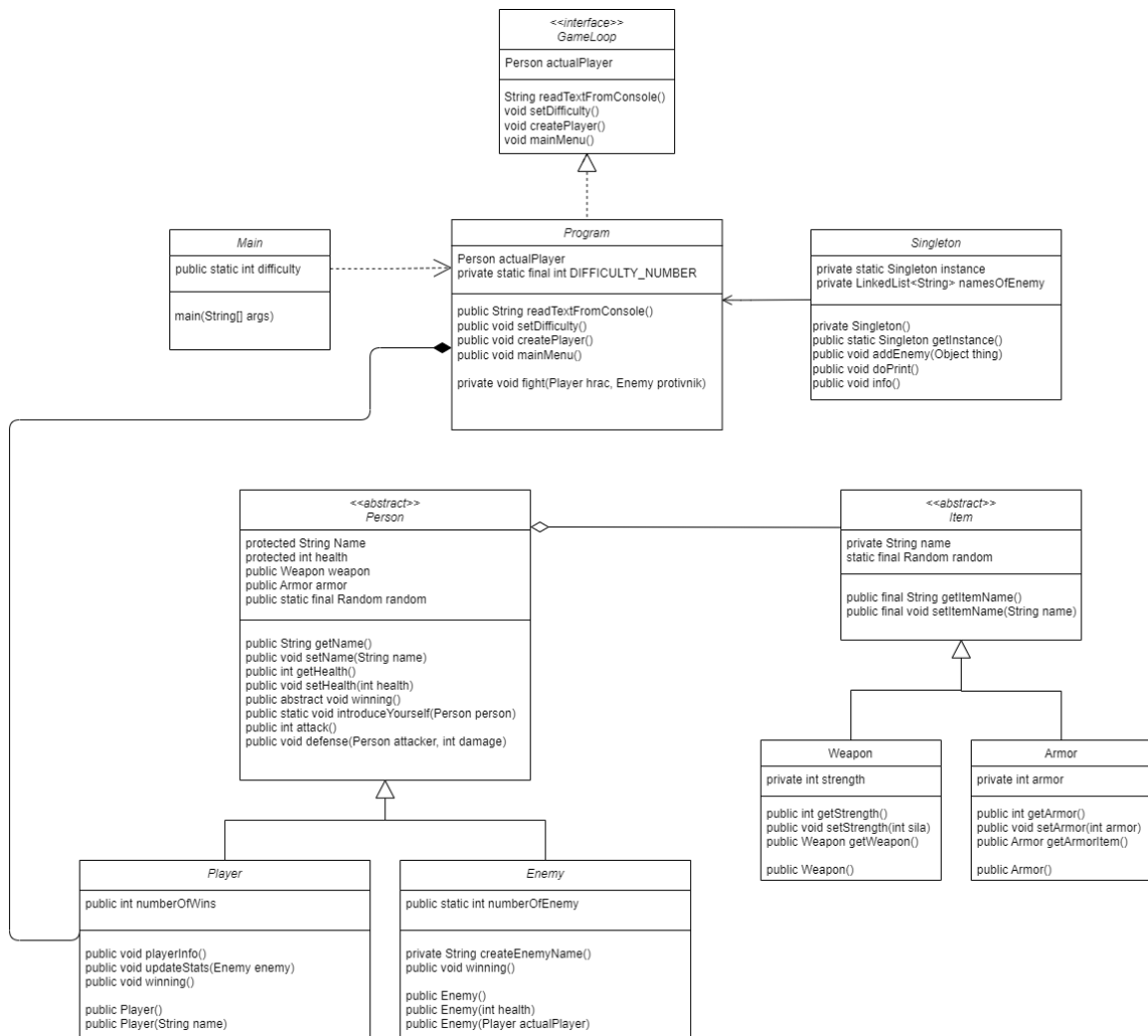
názvy premenných = malými písmenami (camelCase)

názov triedy = prvé veľké písmeno (ProgramLogic, Main...)

metódy = malými písmenami (getName(), winning()...)

názvy, metódy, premenné v anglickom jazyku

24. Vami navrhnutý UML diagram tried (všetky vzťahy a balíky, minimálne 6 tried pre získanie plného počtu bodov). (8)



25. Správne použitie dokumentácie kódu (javadoc + komentare kodu)

(1)

Viac príkladov v kóde, v celom je javadoc a komentáre.

```
FIIT HEROES
Author: Adam Vrabel
Version: ZADANIE 3

6 usages
6 ▶ public class Main {
    6 usages
    7     public static int difficulty = 333;
    8
    9 ▶     public static void main(String[] args) {
        10         System.out.println("ADAM VRABEL, ZADANIE ZOOP");
        11         System.out.println("FIIT HEROES");
        12         System.out.println();
        13
        14         Program run = new Program();
        15
        16         run.setDifficulty();
        17         run.createPlayer();
        18         run.mainMenu();
        19
        20         ///run.trySingleton();
        21         ///run.justTry();
        22     }
    23 }
```

```
Program implementuje INTERFACE GameLoop.
Je to hlavný cyklus hry.
Hra pozostáva z Hráča ( Player ) ktorý bojuje proti protivníkovi ( Enemy )

2 usages
public class Program implements GameLoop {
    16 usages
    Person actualPlayer = null;
    1 usage
    private static final int DIFFICULTY_NUMBER = 15; // KONŠTANTA (final)
    3 usages
    Singleton programDatabase = Singleton.getInstance(); //GET INSTANCE SINGLETON DATABASE

    public void trySingleton(){...}
    public void justTry(){...}

    Funkcia readTextFromConsole() načíta text z konzole.
    Returns: vráti načítaný text (typu String)

    3 usages
    public String readTextFromConsole(){...}

    Funkcia setDifficulty() vypíše možnosti a nastaví náročnosť hry.
    Možnosti náročnosti:
    • ZAČIATOČNÍK
    • POKROČILÝ
```

Funkcia `createPlayer()` vypýta meno z konzoly a vytvorí hráča (`Player`).
Na začiatku hráč dostane zbraň a brnenie. Podľa zvolenej náročnosti hry sa upravuje sila zbrane.

See Also: [items](#),
[characters](#)

1 usage

```
public void createPlayer(){...}
```

Funkcia `mainMenu()` vypíše možnosti hry a umožňuje zvolenie možnosti.

- MOJA POSTAVA
- SUBOJ
- KONIEC

Voľba **MOJA POSTAVA** vypíše informácie o hráčovi (`Player`)

Voľba **SUBOJ** vytvorí protivníka (`Enemy`), jeho život bude v náhodnom okolí hráča (`Player`).
Následne zavolá funkciu `fight()`.

Voľba **KONIEC** ukončí hru a vypíše informácie o hre.

4 usages

```
public void mainMenu(){...}
```

Funkcia `fight()` nasimuluje súboj medzi hráčom (`Player`) a protivníkom (`Enemy`).
Súboj končí pokiaľ jeden z bojovníkov neminie svoj život.

Params: `player` – hráč ktorý bojuje
`enemy` – protivník, ktorý sa bráni a následne útočí hráča

See Also: [Person](#)

1 usage

```
private void fight(Player player, Enemy enemy){...}
```

Návrhový vzor **Singleton** zabezpečí že v programe bude vždy iba jedna inštancia.
Ak ešte inštancia nebola vytvorená tak ju vytvorí.
Ak už inštancia bola vytvorená tak ju iba vráti.

12 usages

```
9 public class Singleton {
    3 usages
10     private static Singleton instance = null;
    1 usage
11     private Singleton(){
    3 usages
12     public static Singleton getInstance(){
13         if(instance == null){
14             instance = new Singleton();
15         }
16         return instance;
17     }
    6 usages
18     private LinkedList<String> namesOfEnemy = new LinkedList<>();
19 }
```

Funkcia `addEnemy(Object thing)` skontroluje či je parameter `Enemy` alebo `String` a priradí do databázy meno bojovníka.

Params: `thing` – typu `Object`

1 usage

```
24 public void addEnemy(Object thing){
25     if(thing instanceof Enemy){
26         namesOfEnemy.push(((Enemy)thing).getName()); // PUSH NAME OF ENEMY TO DATABASE
27     }
```