

## Rapport – Uppgift 3

Daniel Isaksson (Java23)

[Daniel.Isaksson@yh.Nackademin.se](mailto:Daniel.Isaksson@yh.Nackademin.se)

G -

Uppgiften kändes som ett bra sätt att introduceras till objekt. Dock tycker jag att man kan använda sig av getters och setters istället för att skapa en where()-metod. De har exakt samma funktion och det lär ut standarden bättre.

VG -

Jag ville skapa ett program där jag kan simulera korta RPG strider med input från användaren om vad den vill göra. Mycket av det i rapporten skrivs med antagandet att man har en grundläggande koll på gamla RPG-spel. Men jag har försökt försvenska och förklara det på ett tydligt sätt.


Först och främst behövde jag skapa en spelare, så klassen "Player" skapades. Jag ville att spelaren skulle ha följande egenskaper:

- Namn – Vad spelarens karaktär heter
- HP – Liv eller Health Points
- Damage – Hur mycket skada spelaren kan göra
- Gold – Hur mycket guld spelaren har
- Potions – Hur många magiska drycker som återställer liv spelaren har
- Magic – Hur många magiska formler spelaren kan använda sig av (fireball i det här fallet)
- Vapen – Vad spelarens vapen heter

```
1  @ public class Player {
    3 usages
2      private String name;
    3 usages
3      private int health;
    3 usages
4      private int damage;
    3 usages
5      private int gold;
    3 usages
6      private int potions;
    3 usages
7      private int magic;
    3 usages
8      private String weapon;
9
10     > /** Generate a player ...*/
    2 usages
20     public Player(String name, int health, int damage, int gold, int potions, int magic, String weapon) {
21         this.name = name;
22         this.health = health;
23         this.damage = damage;
24         this.gold = gold;
25         this.potions = potions;
26         this.magic = magic;
27         this.weapon = weapon;
28     } //Player
```

Så jag skapade instansvariabler med vad jag behövde och skrev sen en konstruktör med sagda variabler. Skapade därefter getters och setters för allt.

Därefter behövde jag skapa en fiende. Så jag skapade klassen "Enemy" och lät den ärva från Player. Nu när jag skapat en fiende så kan jag utöka klassen "Player" med fler metoder.

```
6 usages
1  public class Enemy extends Player{
2  >  /** Generate a new enemy ...*/
    1 usage
10  public Enemy(String n, int h, int d, int g, int p){
11      super(n, h, d, g, p);
12  } //Enemy
```

Klassen "Player" innehåller nu också två olika sätt för spelaren att skada fienden. Metoderna dealDamage och dealMagicDamage. Skillnaden är att dealDamage tar en tur, medan dealMagicDamage inte gör det. Det är också lite skillnad i vad som printas ut i terminalen.

Jag skapade också metoden win i Player, som används när fiendens liv är 0 eller mindre. Den printar ut lite information om vad spelaren fått för sin seger och uppdaterar därefter spelarens tillhörigheter. Jag är lite anal med grammatik, så la även till en if-sats för potions. Om man får en så skriver den ut i singular, om man får över en skriver den ut i plural.

Enemy-klassen har en egen dealDamage-metod, eftersom utskriften är annorlunda.

Enemy-klassen fick en metod som heter loss. Den skriver ut en text om spelaren hamnar under 0 hp.

Potions och guld i enemy-objektet ges över till spelaren om spelaren besegrar fienden.

Nu kan jag skapa två motståndare. En spelare och en enemy. Med getters och setters kan de påverka varandras variabler.

Därefter behövde jag skapa spelets flöde. Skapade en klass vid namnet "GameLogic". Först ville jag ha en liten textrad för när man stöter på en fiende. Så jag skapade metoden "encounter" och lät den ta in variabler för fiendes attribut. Metoden skriver ut en liten text om fienden, skapar den som ett objekt och skickar objektet till metoden "battle".

Battle är spelets flöde när man slåss. En del if-satser baserat på användar-input bestämmer vad spelet ska göra. Med hjälp av getters och setters så kan programmet hantera spelarens och fiendes agerande. Så som skada, magisk skada eller potions.

```

Welcome!
What is your name?
Daniel
Daniel walks through the forest and a Goblin attacks!
-----
| Daniel 50hp | Goblin 24 hp |
Daniel damage: 10 | potions: 0
What would you like to do? (write number below and press enter)
1: Physical attack | 2: Magical attack | 3: Drink a potion | 4: Help | Q: Quit
1
Daniel deals 10 damage with your stick!
Goblin deals 5 damage!
-----
| Daniel 45hp | Goblin 14 hp |
Daniel damage: 10 | potions: 0
What would you like to do? (write number below and press enter)
1: Physical attack | 2: Magical attack | 3: Drink a potion | 4: Help | Q: Quit
2
You deal 10 damage with your fireball spell!
-----
| Daniel 45hp | Goblin 4 hp |
Daniel damage: 10 | potions: 0
What would you like to do? (write number below and press enter)
1: Physical attack | 2: Magical attack | 3: Drink a potion | 4: Help | Q: Quit
|

```

(Spelets printade flöde)

Jag la även in metoden "fairy", vilken tillåter spelaren att återställa sin HP till max för guld. Än en gång loopar och if-satser för val som användaren gör. Man kan låta bli att ens gå fram till fen, man kan neka att betala eller så kan man betala för tjänsten.

Skapade en Shop-klass för att få nån form av utveckling i karaktären. En konstruktor skapades som tar in variabler för vad vapnet i affären ska heta, hur starkt det ska vara och hur mycket det kostar. Här kan man köpa fler potions eller ett nytt vapen. I det här fallet ett svärd. Mycket av metoden "enterShop" är en loop med printad text och if-satser för val spelaren gör. En liten uppdatering av det som printas sker om spelaren köper ett svärd, för att visa att det redan är sålt.

Därefter ville jag att användaren skulle kunna fortsätta spelet hur länge den vill eller tills karaktären dör. Så jag skapade en "prestige"-mekanik som gör saker svårare efter varje loop av spelet.

Prestige ökar med 1 varje gång spelaren gjort en loop och ökar saker som fiende-skada, fiendens HP, kostnad för vapen och vapens totala skada.

Metoden updateShop i klassen Shop ser till att affären uppdaterar sitt vapen baserat på prestige.

Spelaren kan inte fortsätta i all oändlighet, eftersom fienderna kommer eskalera sin skada i för hög takt för att spelaren ska kunna hålla ikapp. Men ville inte hålla på för mycket med balans i ett program som skulle vara av den enklare varianten.

Till slut skapade jag ett gäng statiska variabler som är till för att skapa objekten player och shop. Användaren får själv bestämma namn på sin player när spelet startas.

För monster kändes det mer flexibelt att använda magiska variabler, även fast det kan ses ner på lite i program. Men i test-syfte lät jag det vara så.

```
while(true){
    GameLogic.encounter(player, print: player.getName() + " walks through the forest", name: "Goblin",
        health: 20 + (prestigeLevel * 4), damage: 4 + prestigeLevel, gold: 10 + (prestigeLevel * 2), potions: 0); <— Skapa fiende
    //encounter with a Goblin

    GameLogic.fairy(player, cost: FAIRY_COST + prestigeLevel); <— Skapa fairy
    //meet a fairy

    GameLogic.encounter(player, print: player.getName() + " comes to a clearing", name: "Ogre",
        health: 40 + (prestigeLevel * 5), damage: 6 + (prestigeLevel * 2), gold: 15 + (prestigeLevel * 3), potions: 1);
    //encounter with an Ogre

    if (GameLogic.prestige(prestigeLevel)){
        shop.enterShop(player, shop); <— Gå in i affär
        prestigeLevel++; <— Uppdatera prestige
        shop.updateShop(shop, prestigeLevel); <— Uppdatera
    } else{                                affär baserat på
        break;                             prestige
    }
}
} //while
```

(While loopen med spelets flöde)

Något jag var lite osäker på är min användning av scanners. Det funkar för programmets syfte, men vet inte om det är optimalt.

Och nu hindrar jag mig själv från att fortsätta lägga till funktionalitet, så att jag inte spenderar all tid jag har till att jobba på ett program som är menat att vara litet. Men jag känner mig ganska nöjd såhär långt.

Jag kanske fortsätter på programmet framåt, för ett annat projekt.