Refaktorointi

Kaspar Tullus

Mitä refaktoroitiin?

Pelaaja.java

Ihan ensimmäiseksi, menin refaktoroimaan pelaajaluokkaa. Siellä refaktoroin "pelaajanValinta" -metodin. Vaihdoin switch-case-rakenteen parempaan vaihtoehtoon, joka on seuraava:

```
public String pelaajanValinta() {
    return new String[]{"kivi", "paperi", "sakset"}[(int) (Math.random() * 3)];
}
```

Tämä toimii samalla tavalla: Math.random() valitsee 0-2 väliltä, ja palauttaa sen perusteella yhden merkkijonon "kivi", "paperi" tai "sakset".

Sitten refaktoroitiin **setVoitot()** ja **getVoitot()** , **setVoitot()** funktio nimettiin **nostaVoittoja**() funktioksi. **SetVoitot()** funktiosta refaktoroitiin seuraavat:

Alkuperäinen:

```
public int setVoitot() {
    int voitotYhteensä = voitot++;
    return voitotYhteensä;
}
```

Refaktoroitu:

```
public void nostaVoittoja() {
    voitot++;
}
```

getVoitot() funktiosta seuraavat:

Alkuperäinen:

```
public int getVoitot() {
    return (voitot);
}
```

Refaktoroitu:

```
public int getVoitot() {
    return voitot;
}
```

Pelaaja luokasta seuraavat muutujat oli myös vaihettu:

Alkuperäinen:

```
public class Pelaaja {
    int voitot;      // Voittojen lukumäärä
    int voitotYhteensä;
```

Refaktoroitu:

```
public class Pelaaja {
private int voitot;
```

Peli.java

Peli-luokassa oli paljon enemmän refaktoroitavaa. Laskut eivät ihan täsmänneet, ja main-koodi oli hyvin sotkuinen ja vaikeasti luettava. Lisäksi huomasin, että siellä oli liikaa if-lauseita ja while-lause pohjalla. Toinen syy, miksi tämä koodi vaati refaktorointia, oli se, että sitä ei voinut testata, koska kaikki oli pakattu yhteen main-funktioon.

Korjasin tilanteen niin, että poistin Peli-luokasta main-funktion ja loin sille oman luokan, jossa käynnistän pelin erikseen:

Main.java

```
/**Main luokka, jossa peliä käynnistetään.*/
public class Main {
    public static void main(String args[]) {
        Pelaaja p1 = new Pelaaja() , p2 = new Pelaaja();
        Peli peli = new Peli(p1,p2);
        peli.pelaa();
    }
}
```

Sitten lähdin tekemään muutoksia Peli-luokkaan. Aloitin korjaamalla kaikki muuttujat private-tyyppisiksi, ja jotkut niistä ovat static final -muuttujia. Syy muuttaa ne privateiksi on, etten halua niiden käytettävän Peli-luokan ulkopuolella:

Alkuperäinen:

```
public class Peli {

    public static void main(String args[]) {
        Pelaaja p1 = new Pelaaja();
        Pelaaja p2 = new Pelaaja();
```

```
        boolean peliLoppui = false;
        int pelatutPelit = 0;           // Pelattujen pelien lkm
        int p1Voitot = p1.voitot;       // Pelaaja 1:n voittojen lkm
        int p2Voitot = p2.voitot;       // Pelaaja 2:n voittojen lkm
        int tasapelit = 0;              // Tasapelien lkm
        String p1Valinta;
        String p2Valinta;
```

Refaktoroitu:

```
public class Peli {
    private static final String KIVI = "kivi";
    private static final String PAPERI = "paperi";
    private static final String SAKSET = "sakset";

    private final Pelaaja p1;
    private final Pelaaja p2;

    private int tasaPelit;
    private int pelatutPelit;

    public Peli(Pelaaja p1, Pelaaja p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
```

Kaiken tämän jälkeen lähdin kirjoittamaan pelaa()-funktiota, jota voisi käyttää main-luokassa:

```
public void pelaa() {
    while (p1.getVoitot() < 3 && p2.getVoitot() < 3) {
        odota();
        pelatutPelit++;
        prosessoiKierros();
        printStatus();
    }
    tulostaVoittaja();
}
```

Tämä vaihtoehto on parempi kuin iso if-lauseiden kirjoittaminen, ja se myös tekee koodista luettavamman!

Sitten kirjoitin prosessoiKierros() function, jonka tehtävänä on pelata pelin kierroksia ja sitten nostaa voittajan pisteitä tai nostaa pelin “tasa-peli” arvoa.

prosessoiKierros() funktio:

```
/**Prosessoi yhden kierroksen ja nostaa voittajan voittoja.*/
private void prosessoiKierros() {
    PeliTulokset result = pelaaYksiKierros(p1.pelaajanValinta(),
p2.pelaajanValinta());
    if (result == PeliTulokset.PELAAJA1_VOITOT) p1.nostaVoittoja();
    else if (result == PeliTulokset.PELAAJA2_VOITOT) p2.nostaVoittoja();
```

```
    else tasaPelit++;
}
```

pelaaYksiKierros() funktio:

```
/**Pelaa yhden kierroksen ja palauttaa voittajan.*/
public PeliTulokset pelaaYksiKierros(String p1Choice, String p2Choice) {
    if (p1Choice.equals(p2Choice)) {
        return PeliTulokset.TASAPELI;
    } else if (p1Choice.equals(KIVI) && p2Choice.equals(SAKSET) ||
            p1Choice.equals(PAPERI) && p2Choice.equals(KIVI) ||
            p1Choice.equals(SAKSET) && p2Choice.equals(PAPERI)) {
        return PeliTulokset.PELAAJA1_VOITOT;
    } else {
        return PeliTulokset.PELAAJA2_VOITOT;
    }
}
```

printStatus() funktio:

```
/**Tulostaa pelin tämän hetkisen tilanteen.*/
private void printStatus() {
    System.out.printf("Erä: %d\nTasapelit: %d\nPelaaja 1 Voittaa:
%d\nPelaaja 2 Voittaa: %d\n",
            getPelatutPelit(), getTasapelit(), p1.getVoitot(),
p2.getVoitot());
}
```

PELAAJA1_VOITOT,TASAPELI ja PELAAJA2_VOITOT tulevat **enum-luokasta**, jonka olen kehittänyt myös tähän tehtävään.

```
/**Enum arvot Pelin tuloksista.*/
public enum PeliTulokset {
    PELAAJA1_VOITOT,
    PELAAJA2_VOITOT,
    TASAPELI
}
```

Peli-luokasta löytyy myös getter-funktioita testeitä varten sekä printStatus-funktiota varten.

```
/**Hakee pelin tasapeli-arvon käynnissä olevasta pelistä*/
public int getTasapelit() {
    return tasaPelit;
}

/**Hakee pelin pelattujen pelien määrän*/
public int getPelatutPelit() {
    return pelatutPelit;
}
```

Testi Luokat

Testejä kirjoitin kaksi kappaletta

- PelaajaTest.java (Testaa Junit5 ja Mockitolla)
- PeliTest.java (Testaa Junit5 ja Mocitolla)

Pelaajatesti-luokassa en ole käyttänyt @ParameterizedTestejä, koska uskon, että niihin ei ollut tarvetta.

Pelitesti-luokassa olen kuitenkin käyttänyt @ParameterizedTestiä, jotta testit eivät olisi liian pitkiä ja ne olisivat luettavassa muodossa:

```
@ParameterizedTest
@CsvSource({
        "kivi, sakset, PELAAJA1_VOITOT",
        "sakset, kivi, PELAAJA2_VOITOT",
        "kivi, kivi, TASAPELI",
        "paperi, kivi, PELAAJA1_VOITOT",
        "kivi, paperi, PELAAJA2_VOITOT",
        "paperi, paperi, TASAPELI",
        "sakset, paperi, PELAAJA1_VOITOT",
        "paperi, sakset, PELAAJA2_VOITOT",
        "sakset, sakset, TASAPELI"
})
public void testYksiKierros(String p1Valinta, String p2Valinta,
PeliTulokset expectedOutcome) {
    PeliTulokset result = peli.pelaaYksiKierros(p1Valinta, p2Valinta);
    assertEquals(expectedOutcome, result);
}
```

Tässä testataan mahdollisimman paljon vaihtoehtoja tuloksille.

Covarage-reportissa on kaikki **100% Methods ja 100% Lines covered jos Main.java ei lasketa mukaan!**