

Kotitehtävä 3 Deadline 28.9. klo 11:45

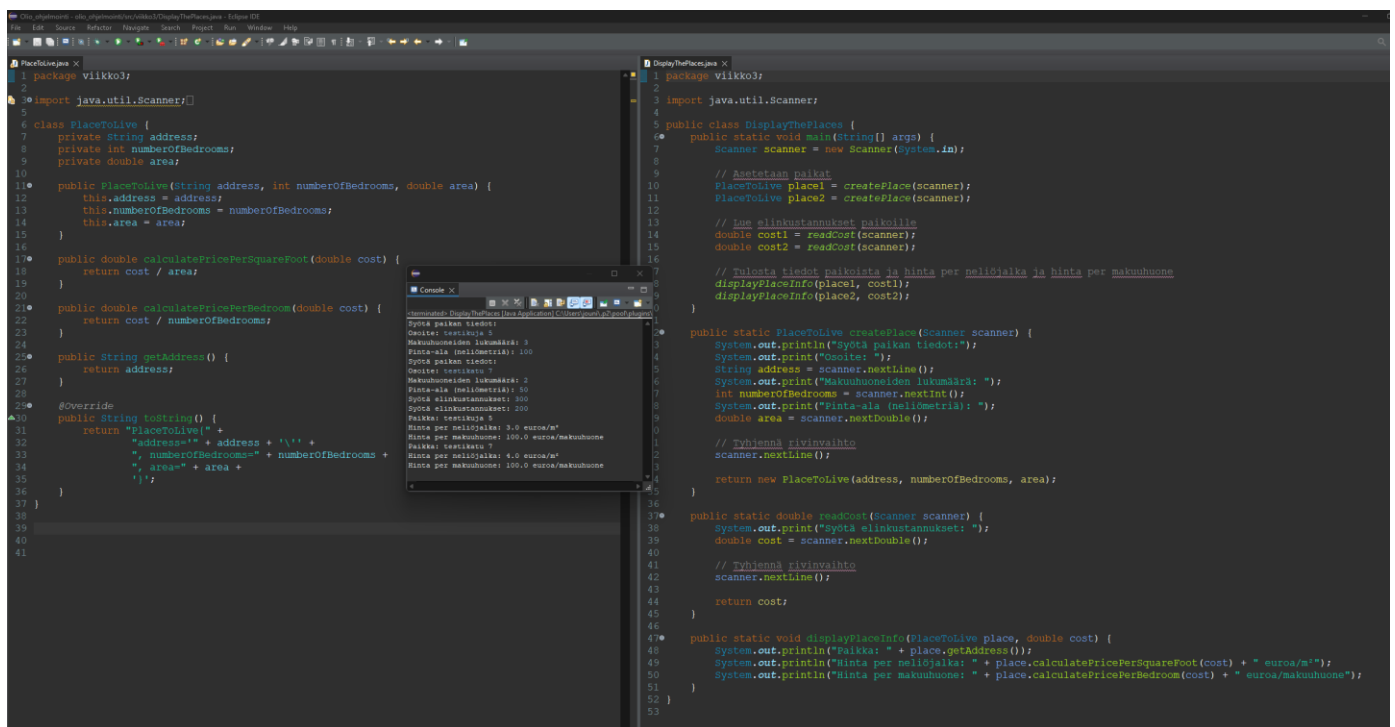
Maksimipisteet 10

Valitse tehtäviä niin, että saat niistä maksimissaan 10 pistettä

Tehtävä 1: Luokat PlaceToLive ja DisplayPlaces (2p)

Locale, Locale, Locale (Try It Out p. 206)

A PlaceToLive has an address, a number of bedrooms, and an area (in square feet or square meters). Write the PlaceToLive class's code. Write code for a separate class named DisplayThePlaces. Your DisplayThePlaces class creates a few PlaceToLive instances by assigning values to their address, numberOfBedrooms, and area fields. The DisplayThePlaces class also reads (from the keyboard) the cost of living in each place. For each place, your code displays the cost per square foot (or square meter) and the cost per bedroom.



```
1 package viikko3;
2
3 import java.util.Scanner;
4
5 class PlaceToLive {
6     private String address;
7     private int numberOfBedrooms;
8     private double area;
9
10    public PlaceToLive(String address, int numberOfBedrooms, double area) {
11        this.address = address;
12        this.numberOfBedrooms = numberOfBedrooms;
13        this.area = area;
14    }
15
16    public double calculatePricePerSquareFoot(double cost) {
17        return cost / area;
18    }
19
20    public double calculatePricePerBedroom(double cost) {
21        return cost / numberOfBedrooms;
22    }
23
24    public String getAddress() {
25        return address;
26    }
27
28    @Override
29    public String toString() {
30        return "PlaceToLive{" +
31            "address='" + address + '\'' +
32            ", numberOfBedrooms=" + numberOfBedrooms +
33            ", area=" + area +
34            '}';
35    }
36 }
37
38
39
40
41
```

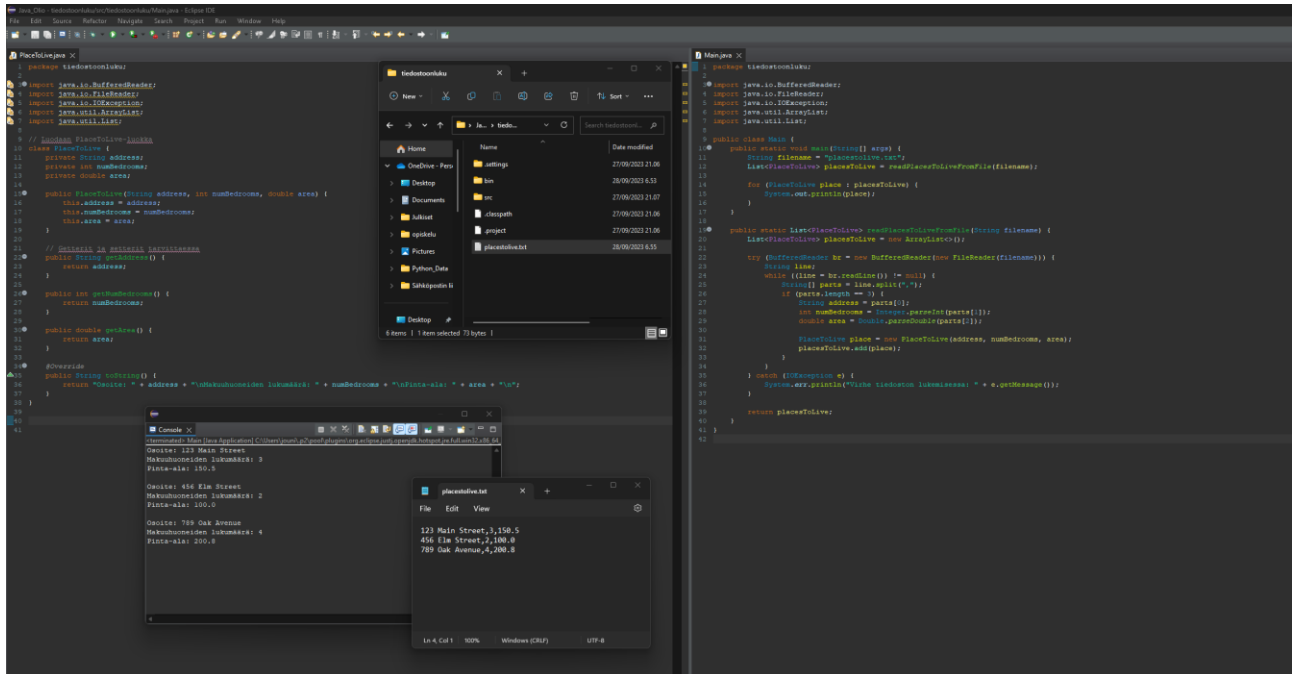
```
1 package viikko3;
2
3 import java.util.Scanner;
4
5 public class DisplayThePlaces {
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8
9         // Luetaan paikat
10        PlaceToLive place1 = createPlace(scanner);
11        PlaceToLive place2 = createPlace(scanner);
12
13        // Lue elinkustannukset paikoille
14        double cost1 = readCost(scanner);
15        double cost2 = readCost(scanner);
16
17        // Tulosta tiedot paikoista ja hinta per neliöalaa ja hinta per makuuhuone
18        displayPlaceInfo(place1, cost1);
19        displayPlaceInfo(place2, cost2);
20    }
21
22    public static PlaceToLive createPlace(Scanner scanner) {
23        System.out.println("Syötä paikan tiedot:");
24        System.out.print("Osoite: ");
25        String address = scanner.nextLine();
26        System.out.print("Makuuhuoneiden lukumäärä: ");
27        int numberOfBedrooms = scanner.nextInt();
28        System.out.print("Pinta-ala (neliömetriä): ");
29        double area = scanner.nextDouble();
30
31        // Tyhjennä rivinvaihto
32        scanner.nextLine();
33
34        return new PlaceToLive(address, numberOfBedrooms, area);
35    }
36
37    public static double readCost(Scanner scanner) {
38        System.out.print("Syötä elinkustannukset: ");
39        double cost = scanner.nextDouble();
40
41        // Tyhjennä rivinvaihto
42        scanner.nextLine();
43
44        return cost;
45    }
46
47    public static void displayPlaceInfo(PlaceToLive place, double cost) {
48        System.out.println("Paikka: " + place.getAddress());
49        System.out.println("Hinta per neliöalaa: " + place.calculatePricePerSquareFoot(cost) + " euroa/m²");
50        System.out.println("Hinta per makuuhuone: " + place.calculatePricePerBedroom(cost) + " euroa/makuuhuone");
51    }
52 }
53
```

```
Syötä paikan tiedot:
Osoite: testitie 5
Makuuhuoneiden lukumäärä: 3
Pinta-ala (neliömetriä): 100
Syötä paikan tiedot:
Osoite: testitie 6
Makuuhuoneiden lukumäärä: 2
Pinta-ala (neliömetriä): 50
Syötä elinkustannukset: 300
Syötä elinkustannukset: 200
Paikka: testitie 5
Hinta per neliöalaa: 3.0 euroa/m²
Hinta per makuuhuone: 100.0 euroa/makuuhuone
Paikka: testitie 6
Hinta per neliöalaa: 4.0 euroa/m²
Hinta per makuuhuone: 200.0 euroa/makuuhuone
```

Tehtävä 2: Tiedoston luku (1p)

On the Record (Try It Out p. 215)

Previously in this chapter, you create instances of your own PlaceToLive class and display information about those instances. Modify the text-based version of your code so that it gets each instance's characteristics (address, number of bedrooms, and area) from a disk file.



Tehtävä 3: Selitä (2p)

Selitä, mihin sanalla extend viitataan, kun luokka määritellään seuraavasti:

```
public class FullTimeEmployee extends Employee { ...
```

miten tämä liittyy termeihin subclass ja superclass/child ja parent?

Sana "extend" viittaa siihen, että luodaan uusi luokka, joka perii (inherit) ominaisuudet ja toiminnallisuuden toiselta luokalta. Tässä tapauksessa luokka "FullTimeEmployee" perii (extends) ominaisuudet ja toiminnallisuuden luokalta "Employee". Tämä tarkoittaa, että "FullTimeEmployee"-luokasta tulee "Employee"-luokan aliluokka (subclass), ja "Employee"-luokasta tulee "FullTimeEmployee"-luokan ylliluokka (superclass) tai vanhempi luokka (parent class).

Terminologiaa selitettynä:

1. **Subclass (aliluokka):** "FullTimeEmployee" on aliluokka, mikä tarkoittaa, että se perii "Employee"-luokan ominaisuudet ja toiminnallisuuden. Aliluokka voi laajentaa ylliluokan toiminnallisuutta tai lisätä siihen uusia ominaisuuksia.
2. **Superclass (ylliluokka) tai Parent Class (vanhempi luokka):** "Employee" on ylliluokka, mikä tarkoittaa, että se antaa perintänä ominaisuudet ja toiminnallisuuden "FullTimeEmployee"-luokalle. Ylliluokka voi toimia yleisenä pohjana useammille aliluokille ja määrittää yhteiset piirteet niille.

Joten kun luokkaa "FullTimeEmployee" määritellään tällä tavalla, se laajentaa (extends) "Employee"-luokan toiminnallisuutta ja voi mahdollisesti lisätä siihen omia erityispiirteitään, jotka ovat ainutlaatuisia tämän aliluokan työntekijöille, mutta se perii kaikki perusominaisuudet ja -toiminnallisuuden "Employee"-luokalta. Tämä mahdollistaa ohjelmoinnissa periytyvyyden käytön ja luokkien hierarkian rakentamisen tehokkaasti ja järjestelmällisesti.

Tehtävä 4: Uudet alaluokat House ja Apartment ja luokan DisplayThePlaces päivitys (2p)

Buy Or Rent (Try It Out p. 224)

Previously in this chapter, you create instances of your own PlaceToLive class and display information about those instances. Create two subclasses of your PlaceToLive class: a House class and an Apartment class. Each House object has a mortgage cost (a monthly amount) and a property tax cost (a yearly amount). Each Apartment object has a rental cost (a monthly amount).

A separate DisplayThePlaces class creates some houses and some apartments. For each house or apartment, your DisplayThePlaces class displays the total cost per square foot (or square meter) and the total cost per bedroom, both calculated monthly.

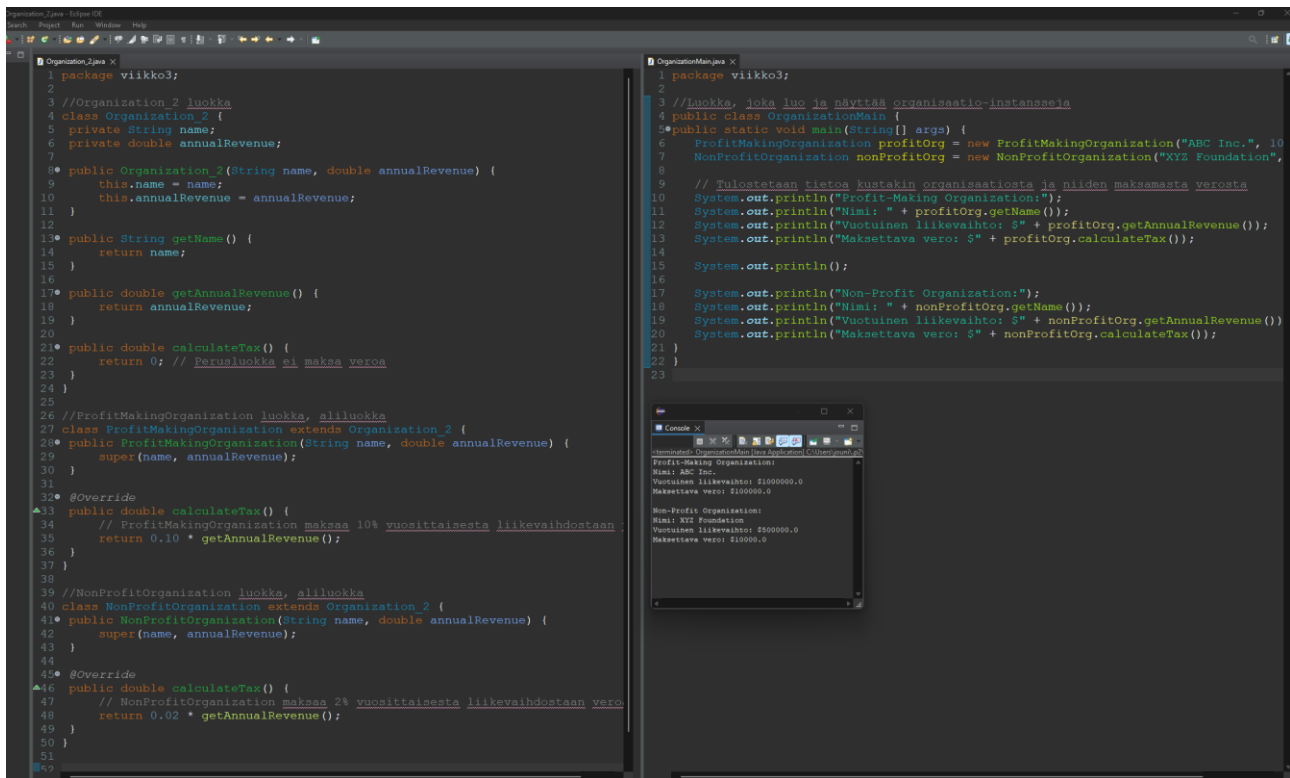
Tehtävä 5: Luokka Organization_2, aliluokat ProfitMakingOrganization ja NonProfitOrganization (2p)

Tax Breaks (Try It Out p. 225)

In Chapter 7, you create an Organization class. Each instance of your Organization class has a name, an annual revenue amount, and a boolean value indicating whether the organization is or is not a profit-making organization.

Create a new Organization_2 class. Each instance of this new class has only a name and an annual revenue amount. Create two subclasses: a ProfitMakingOrganization class and a NonProfitOrganization class. A profit-making organization pays 10 percent of its revenue in tax, but a nonprofit organization pays only 2 percent of its revenue in tax.

Make a separate class that creates ProfitMakingOrganization instances and NonProfitOrganization instances while also displaying information about each instance, including the amount of tax the organization pays.



```
1 package viikko3;
2
3 //Organization 2 luokka
4 class Organization {
5     private String name;
6     private double annualRevenue;
7
8     public Organization(String name, double annualRevenue) {
9         this.name = name;
10        this.annualRevenue = annualRevenue;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public double getAnnualRevenue() {
18        return annualRevenue;
19    }
20
21    public double calculateTax() {
22        return 0; // Perusluokka ei maksa veroa
23    }
24 }
25
26 //ProfitMakingOrganization luokka, allluokka
27 class ProfitMakingOrganization extends Organization {
28     public ProfitMakingOrganization(String name, double annualRevenue) {
29         super(name, annualRevenue);
30     }
31
32     @Override
33     public double calculateTax() {
34         // ProfitMakingOrganization maksaa 10% vuosittaisesta liikevaihdostaan
35         return 0.10 * getAnnualRevenue();
36     }
37 }
38
39 //NonProfitOrganization luokka, allluokka
40 class NonProfitOrganization extends Organization {
41     public NonProfitOrganization(String name, double annualRevenue) {
42         super(name, annualRevenue);
43     }
44
45     @Override
46     public double calculateTax() {
47         // NonProfitOrganization maksaa 2% vuosittaisesta liikevaihdostaan vero
48         return 0.02 * getAnnualRevenue();
49     }
50 }
51
52 }
```

```
1 package viikko3;
2
3 //Luokka, joka luo ja näyttää organisaatio-instanseja
4 public class OrganizationMain {
5     public static void main(String[] args) {
6         ProfitMakingOrganization profitOrg = new ProfitMakingOrganization("ABC Inc.", 1000000);
7         NonProfitOrganization nonProfitOrg = new NonProfitOrganization("XYZ Foundation", 500000);
8
9         // Tulostetaan tietoa kustakin organisaatiosta ja niiden maksamasta verosta
10        System.out.println("Profit-Making Organization:");
11        System.out.println("Nimi: " + profitOrg.getName());
12        System.out.println("Vuosittainen liikevaihto: $" + profitOrg.getAnnualRevenue());
13        System.out.println("Maksettava vero: $" + profitOrg.calculateTax());
14
15        System.out.println();
16
17        System.out.println("Non-Profit Organization:");
18        System.out.println("Nimi: " + nonProfitOrg.getName());
19        System.out.println("Vuosittainen liikevaihto: $" + nonProfitOrg.getAnnualRevenue());
20        System.out.println("Maksettava vero: $" + nonProfitOrg.calculateTax());
21    }
22 }
23 }
```

```
Profit-Making Organization:
Nimi: ABC Inc.
Vuosittainen liikevaihto: $1000000.0
Maksettava vero: $100000.0

Non-Profit Organization:
Nimi: XYZ Foundation
Vuosittainen liikevaihto: $500000.0
Maksettava vero: $10000.0
```

Tehtävä 6: Selitä (3-5p)

Tutustu Java Dokumentaatioista löytyvään tutoriaaliin annotaatioista.

<https://docs.oracle.com/javase/tutorial/java/annotations/> Kerro, mitkä ovat yleisimmät annotaatiot ja mihin niitä käytetään. Anna myös esimerkkejä, miten annotaatioita käytetään. Tehtävää varten ei tarvitse ymmärtää kaikkea, mitä tutoriaalissa sanotaan eikä tarvitse selittää kaikkea.

Jos selitys on kattava (eli menee jo luennotta käydyn asian ulkopuolelle) ja esimerkit oikein hyviä saat 2 lisäpistettä.

Yleisimmät annotaatiot

Javassa on monia erilaisia annotaatioita, mutta tässä kerrotaan muutamista yleisimmistä.

- **@Override**: Tämä annotaatio ilmaisee, että metodin on tarkoitus korvata jokin superluokan metodi.
- **@Deprecated**: Tämä annotaatio ilmoittaa, että metodia tai luokkaa ei pitäisi käyttää enää.
- **@SuppressWarnings**: Tämä annotaatio estää tiettyjen varoitusten, kuten type safety-varoitusten, näyttämisen.
- **@NotNull**: Tämä annotaatio kertoo, että muuttujan tai parametrin tulee olla ei-tyhjä.
- **@Nullable**: Tämä annotaatio kertoo, että muuttujan tai parametrin voi olla myös tyhjä.
- Esimerkkejä annotaatioiden käytöstä

@Override:

```

public class Animal {

    public void makeSound() {
        System.out.println("I make a sound!");
    }
}

public class Dog extends Animal {

    @Override
    public void makeSound() {
        System.out.println("Woof!");
    }
}

```

@Deprecated:

```

public class OldClass {

    @Deprecated
    public void oldMethod() {
        System.out.println("This method is deprecated!");
    }
}

```

@SuppressWarnings:

```

public class SuppressWarningsExample {

    @SuppressWarnings("unchecked")
    public void foo() {
        List<String> list = new ArrayList<>();
        list.add(1); // Tämä aiheuttaa type safety -varoituksen
    }
}

```

@NotNull:

```
public class NullableExample {  
  
    public void foo(@Nullable String name) {  
        if (name != null) {  
            System.out.println(name);  
        }  
    }  
}
```

Annotaatioiden tyypit

Annotaatioita voidaan jakaa kahteen pääluokkaan: meta-annotaatioihin ja muihin annotaatioihin. Meta-annotaatioita käytetään muiden annotaatioiden määrittämiseen. Muita annotaatioita käytetään ohjelman rakenteeseen, käyttäytymiseen tai tietoihin liittyvien tietojen antamiseen.

Meta-annotaatioita on viisi:

1. `@Retention`: Määrittää, kuinka kauan annotaatio on voimassa.
2. `@Target`: Määrittää, mihin elementtiin annotaatio voidaan soveltaa.
3. `@Documented`: Määrittää, tuleeko annotaatio dokumentoida JavaDoc-dokumentaatioon.
4. `@Inherited`: Määrittää, periytyvätkö annotaatiot aliluokille.
5. `@Repeatable`: Määrittää, voidaanko annotaatiota soveltaa samaan elementtiin useita kertoja.

Muita annotaatioita on monia erilaisia, ja niiden käyttötarkoituksia on rajattomasti. Seuraavassa on joitain yleisiä esimerkkejä:

- `@Override`: Ilmaisee, että metodin on tarkoitus korvata jokin superluokan metodi.
- `@Deprecated`: Ilmoittaa, että metodia tai luokkaa ei pitäisi käyttää enää.
- `@SuppressWarnings`: Estää tiettyjen varoitusten, kuten `type safety`-varoitusten, näyttämisen.
- `@NotNull`: Kertoo, että muuttujan tai parametrin tulee olla ei-tyhjä.
- `@Nullable`: Kertoo, että muuttujan tai parametrin voi olla myös tyhjä.

Annotaatioiden käyttöönotto

Annotaatiot lisätään koodiin annotaatiomerkillä. Annotaatiomerkki koostuu annotaation nimestä ja annotaation arvoista. Annotaation nimi on `@`-merkillä alkava sana. Annotaation arvot ovat avain-arvo-pareja, jotka määrittävät annotaation ominaisuuksia.

Esimerkiksi seuraava koodi lisää `@Override`-annotaation `makeSound()`-metodille:

```

public class Animal {
    public void makeSound() {
        System.out.println("I make a sound!");
    }
}

public class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof!");
    }
}

```

Tehtävä 7: Aliluokka ApartmentWithFees (2p)

Pay More and More (Try It Out p. 230)

In previous sections, you create House and Apartment subclasses of your PlaceToLive class. Create an ApartmentWithFees subclass of your Apartment class. In addition to the monthly rental price, someone living in an ApartmentWithFees pays a fixed amount every quarter (every three months). Create a separate class that displays the monthly cost of living in a House instance, an Apartment instance, and an ApartmentWithFees instance.

Tehtävä 8: Vastaa kysymykseen (1p)

Kirjan tehtävä sivu 230, Virtual Methods

Kopioi koodi sivuilta 230-231 ja aja se.

Vastaa seuraavaan kysymykseen:

What output do you see when you run the Main.java file's code? What does this output tell you about variable declarations and method calling in Java?

```

In MySubThing, value is 7
In MyOtherThing, value is 44

```

Tämä tuloste kertoo meille useita asioita muuttujien määrittämisestä ja metodien kutsumisesta Javassa:

1. Polymorfismi: Se, että display-metodia kutsutaan sekä myThing että myThing2 muuttujille ja ne tuottavat erilaisia tulosteita, osoittaa polymorfismia Javassa. Toteutettava metodi riippuu olioiden tyypistä suoritusaikana, ei viittauksen käännösaikaisesta tyypistä.
2. Metodien ylikirjoittaminen: MySubThing- ja MyOtherThing-alaluokat ylikirjoittavat MyThing-yläluokan display-metodin. Tämä mahdollistaa kullekin alaluokalle oman toteutuksen tarjoamisen metodille.

3. Muuttujien julistaminen: Julistukset `MyThing myThing, myThing2;` määrittelevät kaksi muuttujaa, `myThing` ja `myThing2`, tyyppiin `MyThing`. Ne kuitenkin myöhemmin saavat alaluokkien ilmentymät (`MySubThing` ja `MyOtherThing`). Tämä osoittaa, että Javassa voit julistaa muuttujia yläluokan tyyppillä ja sijoittaa niihin alaluokkien ilmentymiä (polymorfismi).
4. Metodin kutsuminen: `display`-metodia kutsutaan sekä `myThing` että `myThing2` muuttujille, mikä näyttää, miten metodeita voidaan kutsua olioihin Javassa. Kutsuttava metodi riippuu varsinaisesta oliotyyppistä, johon muuttuja viittaa suoritusaikana.

Yhteenvetona tämä koodi havainnollistaa keskeisiä Java-ohjelmoinnin periaatteita, mukaan lukien polymorfismi, metodien ylikirjoittaminen, muuttujien julistaminen yläluokilla sekä dynaaminen metodien lähettäminen suoritusaikana