



This Moodle environment is for archive use only. All the active courses at the beginning of the year 2020 are held in the [Learn](#) and [Open Learn](#) environment.

# Oppimistehtävä 4

## Luokat ja metodit

Tee seuraavat tehtävät. Pakkaa tiedostot yhteen .zip -pakettiin ja palauta Moodlen kautta. Kirjoita jokaisen tehtävän ratkaisu niille tarkotettuihin tiedostoihin (ot4a.js, ot4b.js ja ot4c.js). Älä muuta muita koodeja!

Tämän tehtäväsarjan tehtävät koskevat javascript.info:n kappaleita 9.1 - 9.4. Käytä näitä kappaleita materiaalina tehtävien tekemiseen. Voit myös käyttää allaolevaa cheatsheettia tärkeimpien pointtien kertaamiseen.

Mikäli haluat syventyä vielä lisää JavaScript-ohjelmointiin, voit tutustua tehtäväsarjan muihin koodeihin, kuten ot4.html:ään, js/app.js:ään ja ohjelmassa käytettyyn jQuery ja Bootstrap -kirjastoihin ja tutkia miten tämä koko tehtäväsarja on koodattu.

## Cheatsheet

Luokat ovat olio-ohjelmoinnissa keskeisiä rakenteita. Luokkia käytetään monissa muissakin kielissä JavaScriptin lisäksi, joten ohjelmoijan on hyvä tuntea luokkiin liittyvät käsitteet. JavaScriptissa luokat ei kuitenkaan ole yhtä keskeisessä roolissa, kuin esimerkiksi Javassa tai C#:ssa, ja JavaScript-ohjelmointia pystyy harjoittaa myös ilman luokkia. Tällä oppitunnilla otetaan kuitenkin pikainen katsaus luokkiin sen yleishyödyllisyyden vuoksi.

Monipuolinen luokka voi näyttää esimerkiksi tältä:

```

class Luokka1 {
    // Fieldit, eli kentät
    kentta1 = false;
    kentta2 = 'Merkkijono';
    // Privaatti kentät
    _privaKentta1 = 0;
    _privaKentta2 = { arvo: 0, muutosLaskuri: 0 }
    // Propertyt, getterit
    get privaKentta1() {
        return this._privaKentta1;
    }
    get privaKentta2() {
        return this._privaKentta2.arvo;
    }
    // Propertyt, setterit
    set privaKentta2(arvo) {
        this._privaKentta2.arvo = arvo;
        this._privaKentta2.muutosLaskuri = this._privaKentta2.muutosLaskuri + 1;
    }
    // Konstruktori
    constructor(kentta1Alkuarvo) {
        this.kentta1 = kentta1Alkuarvo;
    }
    // Metodit
    metodi1() {
        console.log('Tulostetaan kenttä 2: ' + this.kentta2 + '. Privakenttää muutettu krt: ' + this._privaKentta2.muutosLaskuri);
    }
    summaaKentta1(arvo) {
        return this.kentta1 + arvo;
    }
}

```

Käydään ylläoleva luokka läpi kohta kohdalta:

- Luokan perus määrittely tapahtuu class-syntaksilla **class LuokanNimi { ... }**.
- Koko luokan sisältö kirjoitetaan luokan nimen jälkeen olevien kaarisulkeiden sisään
- Luokan sisällä olevien elementtien järjestyksellä ei ohjelman kannalta ole merkitystä, mutta ne kannattaa selkeyden vuoksi ryhmitellä kenttiin, gettereihin / settereihin, metodeihin yms.
- Luokasta voidaan luoda **instanssi** new-syntaksilla, esim. **new LuokanNimi();**  
Kaarisulkeisiin annetaan konstruktorissa määritellyt parametrit, esim. ylläolevassa esimerkkiluokassa: **const luokka = new Luokka1(true);**
- Luokan konstruktori on funktio, joka kutsutaan luokan luomisvaiheessa. Konstruktori ottaa tavallisten funktioiden tavoin parametreja, joita voidaan käyttää luokan alustamiseen. Esimerkissä konstruktori ottaa vastaan boolean arvon, joka asetetaan kentta1:n arvoksi.
- Luokalla on kenttiä, jotka toimivat luokan sisäisinä muuttujina. Niitä voidaan käsitellä luokan metodeissa this-syntaksilla, kuten: **this.kentta2 = 'Hello world!'**, tai luokan ulkopuolelta, esim: **new Luokka1(true).kentta2 = 'Hello you'**.
- Monissa ohjelmointikielissä luokkien kenttiä voidaan määritellä privaateiksi, jolloin niitä ei voi muuttaa tai lukea luokan ulkopuolisesta koodista, vaan ainoastaan luokan metodeista.

JavaScriptissa kenttiä ei kuitenkaan voida käytännössä määritellä privaateiksi, joten JavaScriptissa on käytäntönä nimetä privaateiksi halutuiksi kenttiä alaviiva-syntaksilla, kuten esimerkkiluokassa esim. **\_privaKentta1 = 0;** Vaikka todellisuudessa näitä kenttiä pystyy muuttamaan myös luokan ulkopuolelta, niin sitä ei pitäisi tehdä, sillä luokan kirjoittaja on tarkoittanut kentän olevan täysin luokan sisäisessä hallinnassa.

- Luokalle voi määritellä properteja, jotka muistuttavat ulkopuolelle kenttiä, mutta niiden käyttöön on liitetty jotakin logiikkaa. Propertyt voivat sisältää gettereitä, settereitä tai molempia. Jos property sisältää vain getterin, kutsutaan sitä read-only propertyksi.
- Ulkopuolelle propertyjä käytetään kenttien tavoin, esim: **const arvo = privaKentta2** kutsuu esimerkkiluokan get privaKentta2() -getterifunktion kutsua, joka palauttaa \_privaKentta2 kenttään tallennetun objektin arvo-kentän (this.\_privaKentta2.arvo). **privaKentta2 = arvo** sen sijaan kutsuu set privaKentta2(arvo) setterifunktiota, joka ei ainoastaan aseta this.\_privaKentta2.arvo:n arvoa, vaan myös kasvattaa \_privaKentta2 objektin muutosLaskurin arvoa. Tämä on vain yksi mielivaltainen esimerkki siitä, minkälaisia toimenpiteitä propertyjen käyttö mahdollistaa. Jos siis kentän muuttamisen pitäisi aiheuttaa luokassa jotakin sivuvaikutuksia, niin kenttien muuttaminen propertyksi voi olla tarpeellista
- **Metodit** ovat luokkaan liitettyjä funktioita, joita käytetään luokkaan liittyvien toiminnallisuuden toteuttamiseen. Metodilla on käytössään luokan kentät ja propertyt, joita metodeista voidaan käyttää this-syntaksilla, esim. **this.\_privaKentta1 = 1000;**
- Metodien määrittelyä **ei** kirjoiteta function-syntaksilla (function metodi1({})), tai lambda-syntaksilla (x => {}), vaan metodien määrittelyyn riittää metodin nimi, jota seuraa sulkeisiin määritellyt parametrit, sekä aaltosulkeisiin kirjoitettu metodin runko, esim.  
**teejotain(param1, param2) { console.log('Tehdään jotain', param1, param2); }** Nämä tietysti kannattaa rivittää hyvien käytäntöjen mukaisesti koodin selkeyttämiseksi.

Luokat siis tarjoavat tavan muodostaa koodissa **abstraktioita**, jotka niputtavat yhteen kuuluvia käsitteitä yhteen. Esimerkki tällaisesta luokasta voisi olla "Ihminen", jolla olisi kenttinä "vasenJalka", "oikeaJalka", "vasenKasi", "oikeaKasi", ja metodeina "kävele" ja "vilkuta". Objekteista luokka eroaa siten, että luokka vain määrittelee kaavion siitä millaisia objekteja luokan ilmentymät (instancsit) ovat. Kun luokasta sitten tehdään instanssi new Luokka() -syntaksilla, synnyttää luokka sen mukaisen objektin, jota koodissa käytetään. Metodit ovat yksinkertaisesti nimitys funktioille, jotka on määritelty luokassa.

Luokat voivat myös periä toisia luokkia, jolloin luokka saa käyttöönsä perityn luokan ominaisuudet. Esimerkiksi, "Ihminen" voisi periä luokan "Nisakas", joka voisi sisältää esim. metodin..... imeta(). Tällöin Ihminen saa automaattisesti käyttöönsä nisäkkään ominaisuudet, mukaanlukien imeta-metodin. Periminen tapahtuu extends-syntaksilla, kuten esim:

```
class Nisakas {  
  imeta() {...}  
}  
class Ihminen extends Nisakas {  
  kavele() {...}  
  vilkuta() {...}  
}
```

Nyt ihminen on määritelty olevan nisäkäs, ja sillä on käytössä imeta-metodi, esim. new Ihminen().imeta(). Keksitkö tapoja jolla ihmiselle ja nisäkkäälle ominaisia toiminnallisuuksia voitaisiin kehittää vielä todenmukaisemmiksi?

Samaa ajatusmallia jatkaen, Nisakas voisi periä luokan "Selkärankainen", joka voisi periä luokan "Selkäjänteinen", joka voisi periä luokan "Eläin", joka voisi periä luokan "Aitotumainen", joka voisi periä luokan "Eliö", joka voisi periä luokan FyysinenObjekti, joka voisi periä luokan

Fysikaalinen Ilmiö jne. Perinnän kautta Ihminen saisi lopulta kaikilta yläluokiltansa suuren määrän ominaisuuksia, joista koostuisi todenmukainen kuvaus ihmiselle ominaisista piirteistä. Näin voisimme teoriassa mallintaa koko maailmankaikkeuden yhdellä tietokoneohjelmalla! Valitettavasti maailmankaikkeuden kaikki informaatio ei toistaiseksi mahdu tietokoneen muistiin ;)

HUOM! Kenttien määrittely suoraa luokan alle allaolevalla syntaksilla ei välttämättä toimi kaikilla selaimilla (Chromella on tuettu):

```
class Luokka {  
    kentta1 = 'arvo1';  
}
```

Ylläolevan syntaksin käyttäminen voi vaatia jotakin kääntäjää, joihin tutustutaan JavaScriptin jatkokurssilla. Jos tämä syntaksi ei toimi sinun selaimellasi, voit alustaa muuttujat konstruktorissa:

```
class Luokka {  
    constructor() {  
        this.kentta1 = 'arvo1';  
    }  
}
```

## Tehtävät

Tehtäviä tehdessäsi pidä huoli, että luokkien, kenttien ja funktioiden nimet ovat tismalleen tehtävänannon mukaisia! Muuten funktioita kutsuva tehtäväkoodi ei toimi oikein.

### a)

Kehitetään edellisen tehtävsarjan tehtävässä 3e tehtyä koodia eteenpäin, ja tehdään kohteesta luokka.

Tee luokka nimeltä "**Kohde**", jonka konstruktori ottaa parametrikseen kohdenumeron, osoitteen, hinnan ja pinta-alan. Tallenna **kohdenumero**, **osoite**, **pintaala** ja **hinta** niin että ne voidaan hakea näillä nimillä luokan ulkopuolelta.

Tee getteri **updated** joka palauttaa false, jos hintaa ei ole muutettu luokan alustuksen jälkeen, ja muuten true.

Tee getteri **neliohint** joka palauttaa kohteen nelihinnan, eli hinta / pinta-ala.

Tee metodi **haeKohde(hakuehto)** joka ottaa parametrina hakuehdon ja palauttaa true, jos kohteen kohdenumero tai osoite sisältää hakuehto-merkkijonon. Tämä on toteutettu funktiona jo edellisen tehtävsarjan tehtävässä 3e, joten voit ottaa sieltä mallia.

Vinkit:

- Koska updated on vain getteri, eikä sitä tule voida muuttaa luokan ulkopuolelta, tulee sinun tehdä esim. \_updated privaattimuuttujaa, jonka palautat getterissä.
- Koska hinnan muuttamisen yhteydessä täytyy luokkaan merkata että hintaa on muutettu, tulee hinta tehdä propertynä, eli määritellä sille getteri ja setteri jotka muuttavat privaattia muuttujaa, esim. \_hint. Näin hinnan setterissä voidaan päivittää updated-propertyä.

- Neliöhinnalle ei tarvitse privaattimuuttujaa, sillä se muodostetaan muista kentistä

Oppimistehtävä 4 A

Lisää kohde

Kohdenumero

Kirjoita tähän...

Osoite

Kirjoita tähän...

Hinta (€)

Kirjoita tähän...

Pinta-ala (m2)

Kirjoita tähän...

Lisää kohde

Hae kohdetta

Hakuehto

Kirjoita tähän...

Hae kohdetta

Kohteet

Kohdenumero	Osoite	Hinta		Pinta-ala	Neliöhinta
123456	Osoitinkatu 2	350000	Päivitä	50	7000.00
55622 (Päivitetty)	Näppiskuja 5	453000	Päivitä	72	6291.67

b)

Tee luokka nimeltä "**Kayttaja**". Käyttäjä ottaa konstruktorissa parametrina vastaa tunnuksen ja salasanan. Tunnus tulee tallentaa luokkaan kenttään nimeltä **tunnus**, mutta salasana tulee tallentaa privaattikenttään, jota ei käytetä luokan ulkopuolelta.

Tee getteri **kirjautunut**, joka palauttaa arvon true, jos käyttäjä on kirjautunut sisään, muussa tapauksessa falsen.

Tee luokalle metodi **kirjauduSisaan(salasana)**, joka ottaa salasanan parametrinaan. Metodin tarkoitus on tutkia onko salasana oikea, ja asettaa luokan sisäinen tila kirjautuneeksi. Onnistuneen kirjautumisen jälkeen kirjautunut-getterin pitäisi palauttaa true. Metodilla on seuraavat ehdot:

- Mikäli käyttäjä on jo kirjautunut sisään, palauttaa metodi: 'Käyttäjä on jo kirjautunut sisään!'
- Mikäli salasana on virheellinen, palauttaa metodi: 'Sisäänkirjautuminen epäonnistui: Väärä salasana!'
- Virheellisen kirjautumisen jälkeen käyttäjän kirjautumiselle asetetaan varoaika, minkä sisällä käyttäjä ei voi yrittää sisäänkirjautumista. Varoaika on aluksi 0 sekuntia, mutta jokaisen peräkkäin tehdyn virheellisen sisäänkirjautumisen jälkeen varoaikaa kasvatetaan 5 sekuntia. Ensimmäisen virheellisen sisäänkirjautumisen jälkeen varoaika on siis 5 sekuntia, ja jos käyttäjä 5 sekunnin jälkeen kirjautuu uudelleen virheellisesti sisään, varoajaksi määrätään 10 sekuntia jne.
- Mikäli käyttäjä yrittää kirjautua varoajan sisällä sisään, palauttaa metodi: 'Sisäänkirjautuminen epäonnistui hetki sitten. Yritä uudelleen X sekunnin päästä', missä X

korvataan varoajan jäljellä olevalla ajalla. Tässä tilanteessa varoaikaa ei siis kasvateta, eikä salasanan vertailua tehdä lainkaan.

- Mikäli varoaikaa ei ole jäljellä ja salasana on oikea, palauttaa metodi: 'Sisäänkirjautuminen onnistui!'. Tämän jälkeen myös kirjautunut-getteri palauttaa true.

Tee luokalle metodi **kirjauduUlos()**, jonka kutsumisen jälkeen kirjautunut-getteri palauttaa false.

Vinkit:

- Varoajan ylläpitämiseksi sinun tulee tehdä privaattikenttä, jonka arvoa kasvatetaan aina 5 sekunnilla epäonnistuneen kirjautumisen yhteydessä
- Saat nykyhetken käyttämällä koodia: **new Date().getTime()**; Tämä antaa nykyhetken millisekunteinä, joten sinun tulee jakaa arvo vielä 1000:lla, jotta voit käsitellä sekunteja. Tallenna edellisen epäonnistuneen kirjautumisyhteyden ajankohta, jotta voit testata uuden yrityksen yhteydessä onko edellisestä yrityksestä kulunut varoajan määrittelemää aikaa.
- Koska kirjautunut-property ei pitäisi päästä muokkaamaan luokan ulkoa, tulee sen tilaa ylläpitää privaattimuuttujassa, esim. `_kirjautunut`, jonka arvo haetaan getterissä. Tarkoitus on, että luokan ulkopuolelta ei voitaisi muuttaa kirjautunut-muuttujan arvoa trueksi, vaan että ainoa tapa kirjautumiseen on käyttää tekemääsi `kirjauduSisaan`-metodia

**Oppimistehtävä 4 B**

### Lisää käyttäjä

Käyttäjänimi

Kirjoita tähän...

Salasana

Kirjoita tähän...

Lisää käyttäjä

Lisättyjä käyttäjiä: 2 (Teemu, Mikko)

### Kirjaudu sisään

Käyttäjänimi

Mikko

Salasana

mallikas

Kirjaudu sisään

Sisäänkirjautuminen onnistui!

Kirjautunut käyttäjä: Mikko

c)

Tee luokka "**Kiinteistovalittaja**", joka perii edellisessä tehtävässäsi tehdyn luokan `Kayttaja`.

Tee luokkaan metodi **luoKohde(kohdenumero, osoite, hinta, pintaala)**, joka ottaa sulkeissa olevat parametrit ja palauttaa a-tehtävässä määritellyn **Kohde**-objektin

Tee luokkaan metodi **paivitaKohde(kohde, hinta)**, joka ottaa parametrinaan a-tehtävässä määritellyn **Kohde**-objektin, sekä hinnan, ja muuttaa kohteen hintaa.

## Lisää käyttäjä

Käyttäjänimi	<input type="text" value="Kirjoita tähän..."/>
Salasana	<input type="text" value="Kirjoita tähän..."/>
<input type="button" value="Lisää käyttäjä"/>	

Lisättyjä käyttäjiä: 2 (Teemu, mikko)

## Kirjaudu sisään

Käyttäjänimi	<input type="text" value="mikko"/>
Salasana	<input type="text" value="mallikas"/>
<input type="button" value="Kirjaudu sisään"/>	

Kirjautunut käyttäjä: mikko

## Lisää kohde

Kohdenumero	<input type="text" value="Kirjoita tähän..."/>
Osoite	<input type="text" value="Kirjoita tähän..."/>
Hinta (€)	<input type="text" value="Kirjoita tähän..."/>
Pinta-ala (m2)	<input type="text" value="Kirjoita tähän..."/>
<input type="button" value="Lisää kohde"/>	

## Kohteet

Kohdenumero	Osoite	Hinta	Pinta-ala	Neliöhinta
Teemu				
12345	Jokikatu 1	<input type="text" value="501000"/> <input type="button" value="Päivitä"/>	28	17892.86
mikko				
99823	Kukkasmäki 95	<input type="text" value="150000"/> <input type="button" value="Päivitä"/>	60	2500.00

## Palautuksen tila

Suorituskerran numero	Tämä on suorituskerta 1
Palautuksen tila	Ei suorituskertoja
Arvioinnin tila	Ei arvioitu
Viimeksi muokattu	-
Palautuksen lisätiedot	► Kommentit (0)

Lisää palautus

Muokkaa palautustasi

## NAVIGOINTI



Työpöytä

Sivuston etusivu

Omat opintojaksoni

Opintojaksokategoriat

Moodle-alustan tilaus (Opettajille)

## ASETUKSET



Opintojakson ylläpito



Oppaat- ja ohjeet - Kysy eTuutorilta - Moodle-tuki