IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

Olio-ohjelmointi: olioiden yhteistyö, vahva kooste

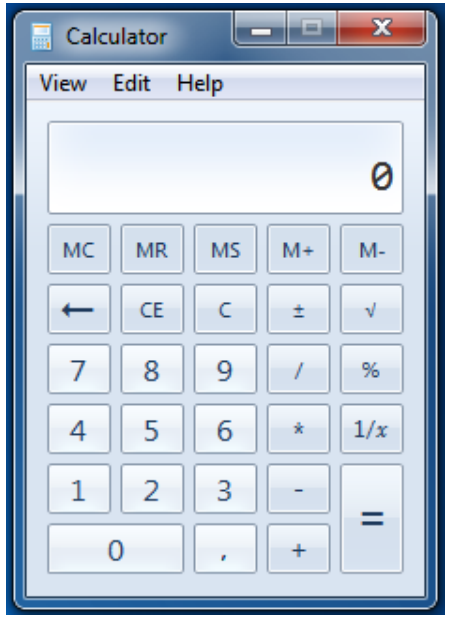
- Olio-ohjelmoinnin yksi perusperiaate on se, että ohjelma rakennetaan toimimaan olioiden **yhteistyöhön** perustuen.
- Tällä opintojaksolla käsitellään luokkien välisistä yhteyksistä kaksi käytetyintä: kooste ja periminen.
- Koosterakenteessa on kysymys omistamisesta. Koosterakenteessa on kooste- ja osaluokka.
- Koosteluokka omistaa osaluokkansa ja on vastuussa näiden käsittelystä.
- Koostumuksen tunnistaa, kun assosiaatioon liittyy sanoja, kuten ”koostuu”, ”sisältää”, ”on osa” jne ; eli sanoja, jotka viittaavat luokkien väliseen koosteiseen suhteeseen.
- Koosteisia suhteita ovat löyhä kooste tai pelkkä kooste (aggregation) ja vahva kooste (composition)

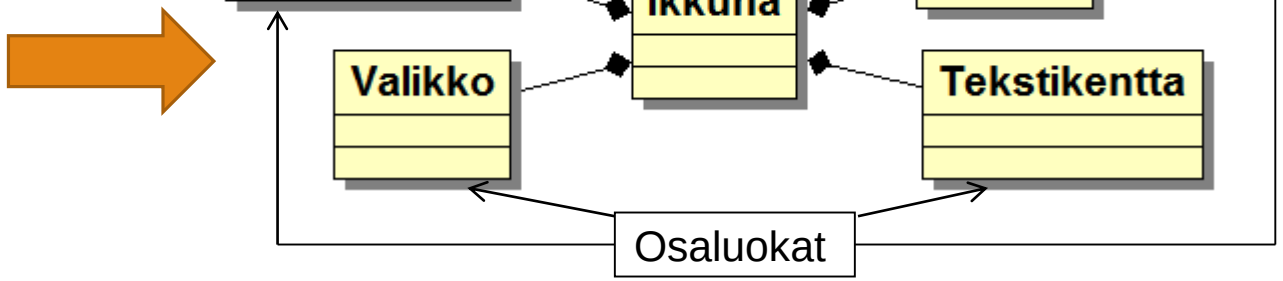
  Composition: Child object dose not have it's own lifecycle and if parent object gets deleted, then all of it's child objects will also be deleted.

- Seuraavissa esimerkeissä käydään läpi vahvaa koostetta.

Olio-ohjelmointi: olioiden yhteistyö, vahva kooste

- Koosteluokka **omistaa** osaluokkansa ja on vastuussa niiden käsittelystä.
- Koska oliot tehdään luokista, niin koosteolion tulee huolehtia osaolion **luonnista** ja **tuhoamisesta.**
- Alla olevassa esimerkissä kun Ikkuna luokasta tehty olio tuhotaan (=ikkuna suljetaan), niin kaikista muistakin ohjelman luokista tehdyt oliot tuhotaan muistista.





Composition: Child object dose not have it's own lifecycle and if parent object gets deleted, then all of it's child objects will also be deleted.

Olio-ohjelmointi: olioiden yhteistyö, vahva kooste

- Luo Qt Creatorilla uusi projekti **MyCompositionProject (**tyyppi **Non-Qt Project>Plain C++ Application)** ja lisää projektiin luokat **MyCompositionClass** (=koosteluokka) ja **MyPartClass** (=osaluokka)
- Esimerkissä koosteluokan tehtävänä on rakentaa vahva kooste luokkien välille.
- Jos luokkien rakenteeseen (.h tiedostot) ei tule automaattisesti muodostin- ja tuhoajafunktiota, niin lisää ne sinne.
- Lisää luokkien toteutukseen (.cpp tiedostot) muodostin- ja tuhoajafunktioiden toteutuksien rungot.
- Lisää luokkien muodostin- ja tuhoajafunktioihin tulostuslauseet **cout** komennolla, esimerkiksi alla olevan mukaisesti:

  cout << "MyCompositionClass muodostinfunktio" << endl;

- Muista lisätä **cout** käskyn/olion tarvitsemat koodirivit **.h** tiedostoihin ennen **class** sanaa!
- Suorita build. *Älä aja ohjelmaa vielä!*

Olio-ohjelmointi: olioiden yhteistyö, vahva kooste

- Vahva kooste voidaan toteuttaa C++ -ohjelmointikielessä kahdella eri tavalla:

  - Osaluokat ovat koosteluokan automaattisia jäsenolioita koosteluokan **private** osassa.

  - Osaluokat toteutetaan dynaamisesti osoittimilla ja olioilla. Oliot voidaan luoda (new) koosteluokan muodostinfunktiossa ja tuhotaan (delete) koosteluokan tuhoojafunktiossa. Myös muissa jäsenfunktioissa voidaan tarpeen vaatiessa hoitaa olioiden luonti ja tuhoaminen.

- Koosteolion tulee huolehtia osaolion **luonnista** ja **hävittämisestä**. Koska tässä esimerkissä on kyseessä on automaattiset oliot, niin käyttöjärjestelmän tehtävänä on huolehtia muistin siivoamisesta. Kooste- ja osaoliot tuhoutuvat muistista **main()** funktion viimeisen rivin suorituksen jälkeen.

Olio-ohjelmointi: olioiden yhteistyö, vahva kooste

Vahva kooste koosteluokan automaattisena jäsenmuuttujana

- Kirjoita **#include "mypartclass.h"** koosteluokan määrittelyyn (**mycompositionclass.h** tiedostoon) ennen **class** sanaa. Jokaisen osaluokan otsikkotiedosto on esiteltävä koosteluokassa.
- Lisää koosteluokan **private** osaan koodirivi **MyPartClass objectMyPartClass; // luodaan automaattinen koosteolio**
- Luo **main()** funktiossa automaattinen olio koosteluokasta.
- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa näytölle? *Huomaa muodostin- ja tuhoajafunktioiden suoritusjärjestys!*
- Siirrä koosteluokan **private** osassa oleva automaattisen olion luontirivi koosteluokan muodostinfunktioon.
- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa näytölle? *Huomaa miten muodostin- ja tuhoajafunktioiden suoritusjärjestys muuttuu!*
- Siirrä automaattisen olion luonti takaisin koosteluokan **private** osaan.

Olio-ohjelmointi: olioiden yhteistyö, vahva kooste

Vahva kooste koosteluokan automaattisena jäsenmuuttujana (jatkuu …)

- Lisää osaluokkaan jäsenfunktio **void myPartClassFunction();** ja lisää jäsenfunktioon tulostuslause

  cout << "Ollaan osaluokan MyPartClass jasenfunktiossa myPartClassFunction()" << endl;

- Lisää koosteluokan muodostinfunktioon alla oleva koodirivi

  objectMyPartClass.myPartClassFunction(); // kutsutaan osaluokan jasenfunktiota

- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa näytölle?

- ***Koosteluokasta voidaan kutsua siis osaluokan olion avulla osaluokan jäsenfunktiota --> tämä mahdollistaa olioiden yhteistyön.***

Olio-ohjelmointi: olioiden yhteistyö, vahva kooste

Vahva kooste

- Lisää osaluokan **private** osaan jäsenmuuttuja **short myPartClassVariable;** - anna jäsenmuuttujalle alkuarvoksi 15 osaluokan muodostinfunktiossa.

- Lisää osaluokkaan jäsenfunktio **short mySecondPartClassFunction();** ja lisää jäsenfunktioon koodirivi

  return myPartClassVariable;

- Lisää koosteluokan muodostinfunktioon alla oleva koodirivi

  cout << "Osaluokan jasenmuuttuja myPartClassVariable = "<< objectMyPartClass. mySecondPartClassFunction() << endl;

- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa näytölle?

- ***Osaluokasta voidaan palauttaa jäsenmuuttujan arvo koosteluokalle -> mahdollistaa olioiden yhteistyön.***

Olio-ohjelmointi: olioiden yhteistyö, vahva kooste dynaamisen muistinhallinnan mukaisesti

- Kommentoi koosteluokan **private** osassa oleva koodirivi **MyPartClass objectMyPartClass;**
- Lisää koosteluokan **private** osaan koodirivi: **MyPartClass *objectMyPartClass; // luodaan osoitin joka on osaluokan tyyppinen**

- Kommentoi koosteluokan muodostinfunktiossa olevat seuraavat //koodirivit

```
objectMyPartClass.myPartClassFunction();
cout << "Osaluokan jasenmuuttuja myPartClassVariable = "<< objectMyPartClass.mySecondPartClassFunction() << endl;
```

- Kirjoita koosteluokan muodostinfunktioon koodirivi **objectMyPartClass = new MyPartClass; // luodaan osaolio**

- Lisää koosteluokan tuhoajafunktioon alla olevat koodirivit

```
delete objectMyPartClass; // koosteolio on vastuussa osaolion/osaolioiden tuhoamisesta muistissa
objectMyPartClass = nullptr;
```

- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa näytölle? Huomaa muodostin- ja tuhoajafunktioiden suoritusjärjestys.

Olio-ohjelmointi: olioiden yhteistyö, vahva kooste dynaamisen muistinhallinnan mukaisesti

- Kirjoita koosteluokan muodostinfunktioon olion luonnin jälkeen seuraavat koodirivit. Huomaa, että jäsenfunktioita kutsutaan **->** operaattorin avulla, koska kyseessä dynaamisesti luotu olio.

```
objectMyPartClass->myPartClassFunction();
cout << "Osaluokan jasenmuuttuja myPartClassVariable = "<< objectMyPartClass->mySecondPartClassFunction() << endl;
```

- Suorita build ja aja ohjelmaa. Mitä ohjelma tulostaa näytölle?