IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

LISÄTEHTÄVÄ OSA 4

SISÄLTÖ

1. Yhteyssuhde (association)
2. Vahva kooste (composition)

LISÄTEHTÄVÄ OSA 4

- **Yhteyssuhde (association)**

- Olioiden välillä voi olla tilannekohtaisia yhteyssuhteita, kun eri luokkien olioiden täytyy suorittaa jotain tehtävää sovelluksessa **yhdessä**.
- Yhteyssuhde toteutetaan osoitinmuuttujien tai viittausten avulla, tai tarvittaessa erillisellä yhteyssuhde luokalla.
- Yksisuuntaisessa (uni-directional association) assosiaatiossa viestintä voi tapahtua vain yhteen suuntaan, ja kaksisuuntaisessa (bi-directional association) viestintä voi tapahtua molempiin suuntiin.
- Yhteyssuhteen tunnistaa luokkien välillä siitä, että kyseessä **ei ole** erityistä omistussuhdetta luokkien välillä, kuten esimerkiksi vahvassa koosteessa tällainen on.

- **Vahva kooste (composition)**

- Koosteluokka omistaa osaluokkansa ja on vastuussa näiden käsittelystä (=koosteolio huolehtii osaolion luomisesta ja tuhoamisesta).
- Osaolio ei ole olemassa ilman koosteoliota ja osaolio voi olla vain yhden koosteolion osa.
- Koostumuksen tunnistaa, kun assosiaatioon liittyy sanoja, kuten ”koostuu”, ”sisältää”, ”on osa” jne ; eli sanoja, jotka viittaavat luokkien väliseen koosteiseen suhteeseen.
- Vahvan koosterakenteen yksi olennainen piirre on se, että osaluokat häviävät, kun koosteluokka häviää. Tämän piirteen avulla voi tunnistaa, onko kyse vahvasta koosteesta.

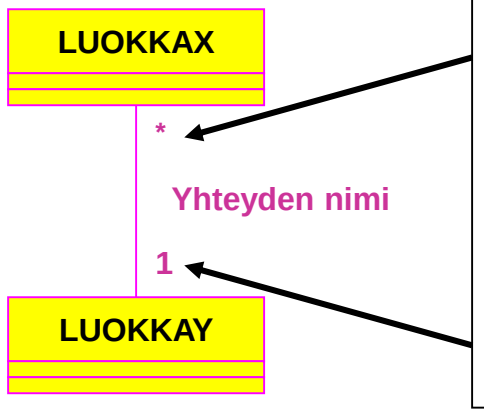
1. Yhteyssuhde

- Yhteyssuhde eli assosiaatio on yhteys luokkien välillä.

- Luokkien välillä voi olla seuraavia yhteyssuhteita.

  - Yhden suhde yhteen 1:1,yhden suhde moneen 1:M tai monen suhde moneen M:N
    (Huom! Vertaa tietokannan taulujen yhteyksiin)

- Yhteyssuhteessa luokkien välillä kerrotaan yhteyden nimi ja kerrannaisuus.

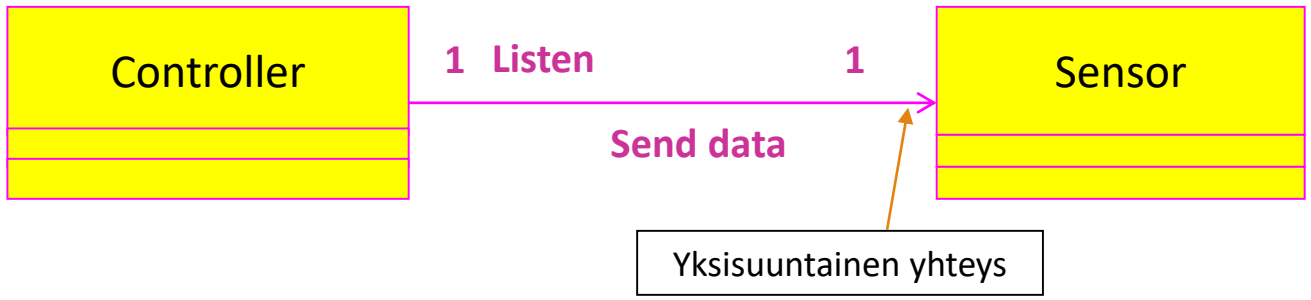


Kerrannaisuus voidaan ilmaista seuraavasti
- 1 tarkoittaa yhtä yhteyttä luokkien välillä
- * tarkoittaa määrittelemättömän monta
- 0..1 tarkoittaa kertautumista "yksi tai ei yhtään"
- 0..* tarkoittaa kertautumista "ei yhtään tai määrittelemättömän monta"
- Lukuarvot voidaan ilmaista myös tarkemmin esim.
  5..100

Yhden suhde yhteen (1:1), yksisuuntainen yhteys

- Yksi Controller luokka on yhteydessä vain yhteen Sensor luokkaan.

- Controller luokan olio kuuntelee (listen) Sensori luokan oliota
- Sensor luokan olio lähettää (Send data) dataa vain yhdelle Controller luokan oliolle.
- Yhteyssuhde on tyypiltään **yksisuuntainen**, joten koodissa Controller luokkaan tulee **private** osaan osoitin, joka on luokan Sensor tyyppinen.

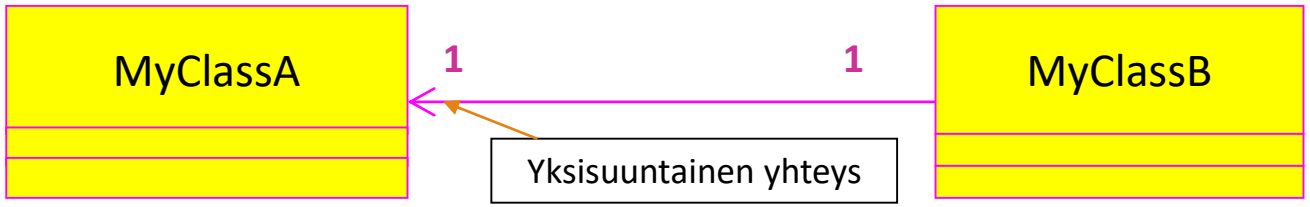



```
#include "sensor.h"
class Controller
{
private:
  Sensor* objectSensor;
};
```

Esimerkki: Yhden suhde yhteen (1:1), yksisuuntainen yhteys

- Yksi **MyClassA** luokka on yhteydessä vain yhteen **MyClassB** luokkaan.

- Yhteyssuhde on tyypiltään **yksisuuntainen**, joten koodissa **MyClassB** luokkaan tulee **private** osaan osoitin, joka on luokan **MyClassA** tyyppinen.
- Yhteys muodostetaan **MyClassB** luokan muodostinfunktiossa




```
// main.cpp
#include "myclassa.h"
#include "myclassb.h"

int main()
{
   MyClassA *a = new MyClassA;
   MyClassB *b = new MyClassB(a);

   delete a;
   delete b;
   a = nullptr;
   b = nullptr;

   return 0;
}
```

```
// myclassa.h
#include <iostream>
using namespace std;

class MyClassA
{
public:
   void functionA();
};
```

```
// myclassb.h
#include "myclassa.h"

class MyClassB
{
public:
   MyClassB(MyClassA *pA);
private:
   MyClassA *objectA;
};
```

```
// myclassa.cpp
#include "myclassa.h"

void MyClassA::functionA()
{
   cout << "fA()" << endl;
}
```

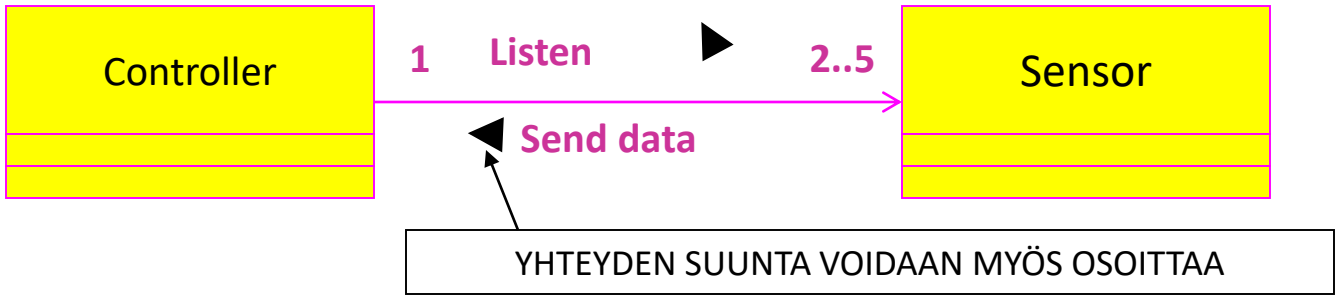
```
// myclassb.cpp
#include "myclassb.h"
#include "myclassa.h"

MyClassB::MyClassB(MyClassA *pA)
{
   objectA = pA;
   objectA->functionA();
}
```

Yhden suhde moneen (1:N)

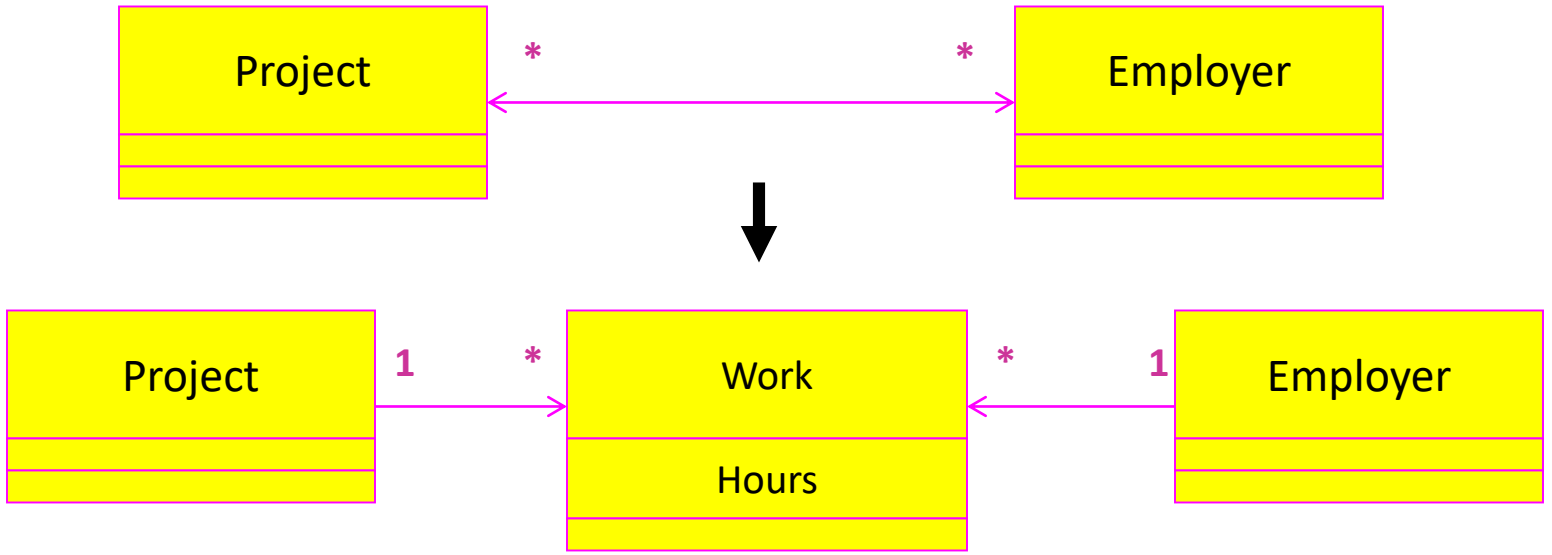
- Yrityksessä työskentelee **monta** työntekijää ja tietty henkilö työskentelee vain **yhdessä** yrityksessä

- Kontrolleri kuuntelee viestejä **2-5**:ltä sensorilta ja sensorit lähettävät viestejä vain **yhdelle** kontrollerille



Monen suhde moneen (N:M)

- Kaksisuuntainen N:M -yhteys muutetaan kahdeksi 1:M –yhteydeksi
- Projektissa työskentelee **monta** työntekijää ja työntekijä työskentelee **monessa** projektissa

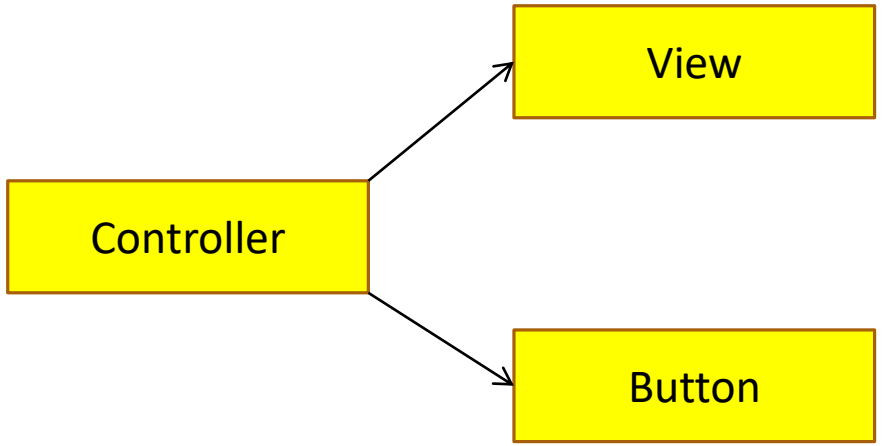


- **Work** luokan avulla saadaan selville esim. työntekijöiden tekemät tuntimäärät eri projekteille

Olio-ohjelmointi: Esimerkkitehtävä, olioiden yhteistyö (yhteyssuhde)

- Tässä tehtävässä pitää rakentaa kolmen luokan avulla ohjelma, joka lukee käyttäjältä merkkejä näppäimistöltä ja tulostaa näppäimistöltä painetun merkin näytölle. Anna projektille nimeksi **Yhteyssuhde** (**Non-Qt Project->Plain C++ Application**) .

- Ohjelmaan pitää suunnitella ja toteuttaa kolme luokkaa:



- Controller –luokka. Luokan tehtävänä on kuunnella tuleeko viesti näppäimistöltä, jos jotain näppäintä on painettu (toteutetaan funktiolla **kbhit()**, joka kuuluu kirjastoon **conio.h**). Kun käyttäjä on painanut jotain näppäintä, niin Controller luokan olio pyytää Button luokan oliota lukemaan merkin näppäinpuskurista, ja sen jälkeen Controller luokan olio pyytää View luokan oliota tulostamaan merkin näytölle. ***Tämä on olioiden yhteistyötä!***

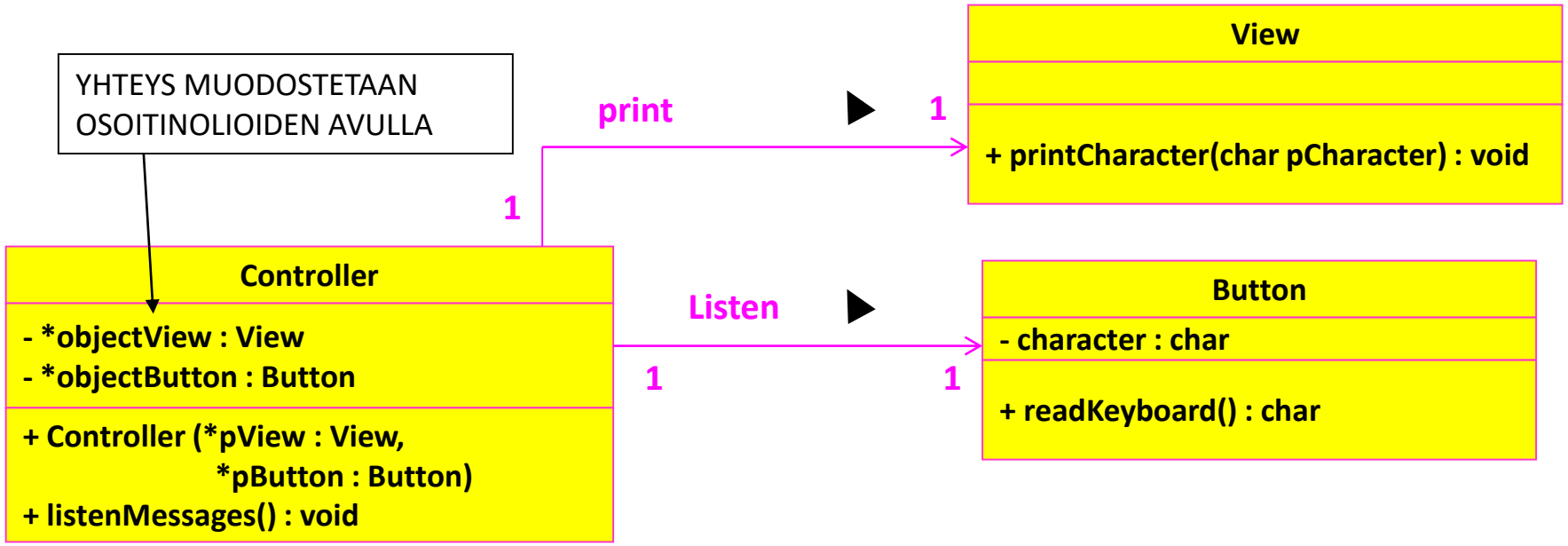
- View –luokka. Luokan tehtävänä on tulostaa näppäimistöltä painettu merkki näytölle (toteutetaan funktiolla **printf()**, kuuluu kirjastoon **stdio.h**).

- Button –luokka. Luokan tehtävänä on lukea näppäimistöltä painettu merkki näppäinpuskurista (toteutetaan funktiolla **getch()**, kuuluu kirjastoon **conio.h**).

Lisää ohjeita seuraavalla sivulla!

Esimerkkitehtävä: Yhteyssuhde 1:1

- Lisää luokat projektiin aikaisemmin opitun mukaisesti, jolloin myös #ifndef, #define ja #endif rivit tulevat luokan määrittelyyn mukaan.
- Luokkien määrittelyt (.h tiedostot) seuraavalla sivulla.



Esimerkkitehtävä: Yhteyssuhde 1:1

- Alla luokkien C++ koodi.

controller.h

```
#ifndef CONTROLLER_H
#define CONTROLLER_H

#include "view.h"
#include "button.h"

class Controller
{

private:
   View *objectView;
   Button *objectButton;

public:
   Controller(View *pView,Button *pButton);
   void listenMessages();
};

#endif // CONTROLLER_H
```

view.h

```
#ifndef VIEW_H
#define VIEW_H

#include <stdio.h>

class View
{
public:
   void printCharacter(char pCharacter);
};

#endif // VIEW_H
```

button.h

```
#ifndef BUTTON_H
#define BUTTON_H

#include <conio.h>

class Button
{
private:
   char character;
public:
   char readKeyboard();
};

#endif // BUTTON_H
```

Esimerkkitehtävä: Yhteyssuhde 1:1. Yhteyssuhde ohjelmakoodissa.

- Toteuta ohjelman main() -funktio ja Controller luokan jäsenfunktiot alla olevan mukaisesti.
- Yhteyssuhde 1:1 muodostetaan ohjelman main() -funktiossa

```
#include "view.h"
#include "button.h"
#include "controller.h"

void main()
{
   View *objView = new View;
   Button *objButton = new Button;
   Controller *objController;

  // Yhteyssuhteen 1:1 muodostaminen alla olevalla rivillä
  objController = new Controller(objView, objButton);
 // Ylläolevan olion luonti aiheuttaa muodostinfunktion suorittamisen

  delete objView;
  delete objButton;
  delete objController;

  objButton = nullptr;
  objView = nullptr;
  objController = nullptr;

}
```

```
Controller::Controller (View *pView, Button *pButton)
{
   objectView = pView;
   objectButton = pButton;
   this->listenMessages();
}
```

Lisää ohjeita seuraavalla sivulla!

```
void Controller::listenMessages()
{
   char keyPressed='0';

   while (keyPressed!='q')
   {
      // if (kbhit()) ehtolause toteutuu, jos nappainta painetaan
      if (kbhit())
      {
         //alla koodi, jossa oliot hoitavat merkin lukemisen ja tulostamisen
         keyPressed=objectButton->readKeyboard();
         objectView->printCharacter(keyPressed);
      }
   }
}
```

Esimerkkitehtävä: Yhteyssuhde 1:1. Yhteyssuhde ohjelmakoodissa.

- Toteuta ohjelman Button ja View luokan jäsenfunktiot alla olevan mukaisesti.

```
#include "button.h"

char Button::readKeyboard()
{
   character = getch();
   return keyPressed;
}
```

```
#include "view.h"

void View::printCharacter(char pCharacter)
{
   printf("%c", pCharacter);
}
```

- Suorita build ja testaa ohjelmaa.

- Jos koodiriville **character = getch();** työkalu antaa varoituksen, niin se johtuu siitä, että getch() –funktio palauttaa int tietotyyppiä määrittelyn mukaisesti. Ohjelma toimii silti halutun mukaisesti.
- Jos haluat päästä varoituksesta eroon, niin täytyy tehdä tyyppimuunnos (type conversion) alla olevan mukaisesti: **character = char(getch());**

- Suorita build ja testaa ohjelmaa.

2. Kooste –suhde

- Koostumussuhde on assosiaation erikoismuoto.

- Koosterakenteessa on kysymys **omistamisesta**.

- Koosteluokka **omistaa** osaluokkansa ja on **vastuussa** näiden käsittelystä (=koosteolio huolehtii osaolion luomisesta ja tuhoamisesta).

- Osaolio ei ole olemassa ilman koosteoliota ja osaolio voi olla vain yhden koosteolion osa.

- Koostumuksen tunnistaa, kun assosiaatioon liittyy sanoja, kuten ”koostuu”, ”sisältää”, ”on osa” jne ; eli sanoja, jotka viittaavat luokkien väliseen koosteiseen suhteeseen.

- Koosteisia suhteita ovat **aggregation** (löyhä kooste tai pelkkä kooste) ja **composition** (vahva kooste).

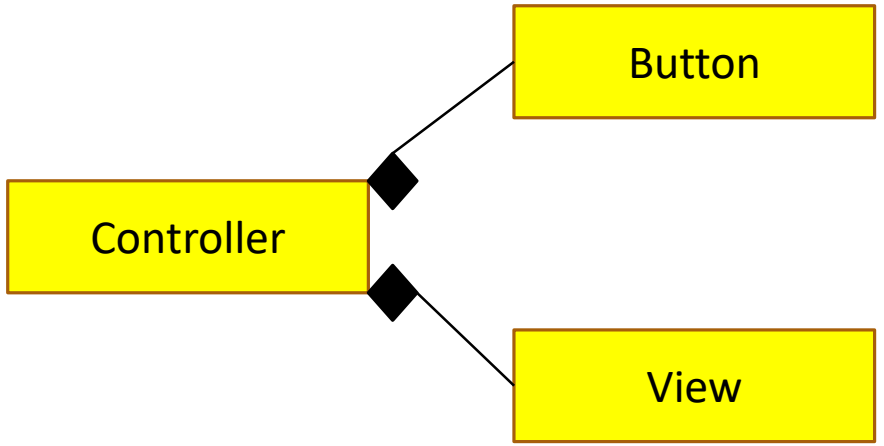
Olio-ohjelmointi: Lisätehtävä 1, olioiden yhteistyö (vahva kooste)

- Tässä tehtävässä pitää rakentaa kolmen luokan avulla ohjelma, joka lukee käyttäjältä merkkejä näppäimistöltä ja tulostaa näppäimistöltä painetun merkin näytölle. Anna projektille nimeksi **VahvaKooste** (**Non-Qt Project->Plain C++ Application**) .

- Ohjelmaan pitää suunnitella ja toteuttaa kolme luokkaa:
  - Controller –luokka. Tämä luokka on **koosteluokka**. Sen tehtävänä on kuunnella tuleeko viesti näppäimistöltä, jos jotain näppäintä on painettu (toteutetaan funktiolla **kbhit()**, kuuluu kirjastoon **conio.h**). Kun käyttäjä on painanut jotain näppäintä, niin Controller luokan olio pyytää Button luokan oliota lukemaan merkin näppäinpuskurista, ja sen jälkeen Controller luokan olio pyytää View luokan oliota tulostamaan merkin näytölle. ***Tämä on olioiden yhteistyötä!***

  - Button –luokka. Tämä luokka on **osaluokka**. Luokan tehtävänä on lukea näppäimistöltä painettu merkki näppäinpuskurista (toteutetaan funktiolla **getch()**, kuuluu kirjastoon **conio.h**).

  - View –luokka. Tämä luokka on **osaluokka**. Luokan tehtävänä on tulostaa näppäimistöltä painettu merkki näytölle (toteutetaan funktiolla **printf()**, kuuluu kirjastoon **stdio.h**).



Lisää ohjeita seuraavalla sivulla!

Olio-ohjelmointi: Lisätehtävä 1, olioiden yhteistyö (vahva kooste)

- Tehtävässä pitää käyttää osoittimia ja dynaamista muistinhallintaa.
- Controller luokassa pitää olla muodostin- ja tuhoajafunktiot, joissa osaoliot luodaan ja tuhotaan vahvan kooste – yhteyden mukaisesti.
- View ja Button luokissa ei tarvitse olla muodostin- ja tuhoajafunktioita, koska niillä ei ole mitään tehtävää ohjelmassa.
- **main()** –funktiossa luodaan ja tuhotaan Controller luokan olio.
- Jos käyttäjä painaa merkkiä ’q’ , niin ohjelma lopetetaan ja oliot tuhotaan muistista.

Olio-ohjelmointi: Lisätehtävä 1, olioiden yhteistyö (vahva kooste)

- Ohjelman toimiva ratkaisu pitäisi saada koodirivien määrän suhteen – pois lukien tyhjät rivit - tehtyä seuraavasti:

  - main.cpp, 8 koodiriviä
  - controller.h, 16 koodiriviä
  - view.h, 9 koodiriviä
  - button.h, 11 koodiriviä
  - controller.cpp, 26 koodiriviä
  - view.cpp, 5 koodiriviä
  - button.cpp, 6 koodiriviä

- Koodirivien optimi määrä ei ole tässä tehtävässä se olennaisin asia vaan se, että asia tulee ymmärrettyä ja ratkaisu tehtävään tulee tehdyksi.

- ***Vertaa Esimerkkitehtävän ja tehtävän 1 ratkaisuja, ja tunnista erot!***