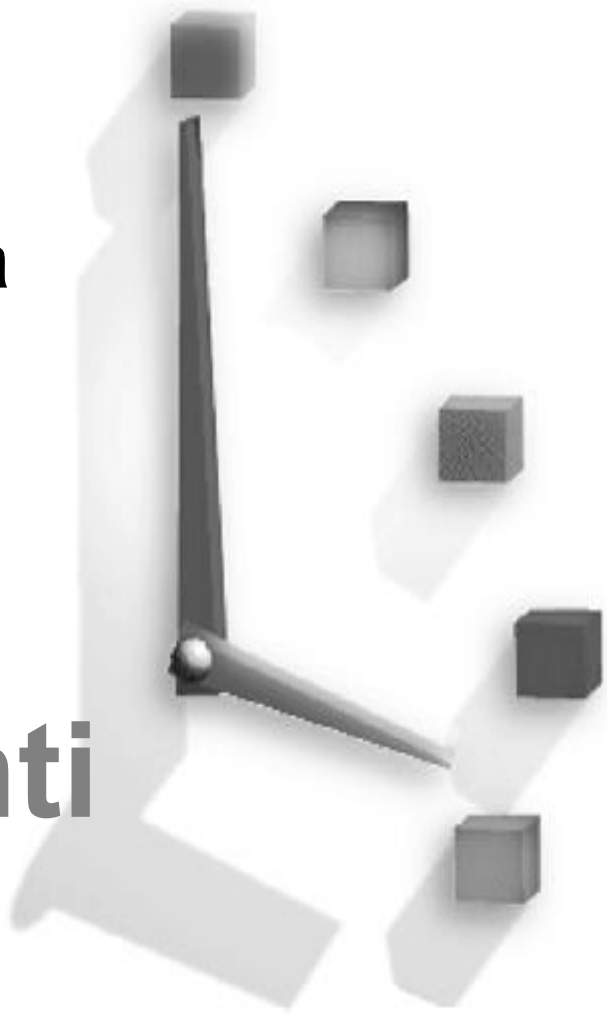
Osa
II



7. oppitunti

Lisää luokista

Aiemmassa luvussa opit luomaan uusia tietotyyppejä esittelemällä luokkia. Tässä luvussa opit hallitsemaan luokkia ja käyttämään kääntäjää auttamaan virheiden löytämisessä ja välttämisessä. Luvun aiheita ovat

- Mitä ovat vakiot jäsenfunktiot
- Kuinka erotetaan luokan käyttöliittymä sen toteutuksesta

Vakiomuotoiset jäsenfunktiot

Uusi käsite Jos esittelet jäsenfunktion vakiona, päätät samalla, ettei tuo jäsenfunktio muuta minkään luokan jäsenen arvoa. Vakiona esittely tapahtuu varatulla sanalla const, joka sijoitetaan sulkumerkkien jälkeen ennen puolipistettä.

Tällainen funktio ei ota argumentteja ja palauttaa void-arvon. Se näyttää seuraavanlaiselta:

void SomeFunction() const;

Käyttöfunktiot esitellään usein vakioina const-lisämääreen avulla. Cat-luokalla on kaksi käyttöfunktiota:

```
void SetAge(int anAge);
int getAge();
```

SetAge() ei voi olla vakio, koska se muuttaa jäsenmuuttujaa itsAge. Sen sijaan GetAge() voi ja sen pitäisi olla vakio, koska se ei muuta luokkaa lainkaan. Se vain palauttaa jäsenmuuttujan itsAge nykyisen arvon. Siksi näiden funktioiden esittelyn tulisi olla seuraavanlainen:

```
void SetAge(int anAge);
int GetAge() const;
```

Jos funktio on esitelty vakiona ja toteutus yrittää muuttaa kohdetta (muuttamalla jäsenen arvoa), ilmoittaa kääntäjä virheestä. Jos kirjoitat esimerkiksi funktion GetAge() siten, että se laskee Cat-olion iän kysymisten määrän, antaisi kääntäjä virheilmoituksen. Se johtuu siitä, että tällöin Cat-oliota muutettaisiin metodia kutsuttaessa.

Huom! Käytä const-avainsanaa aina kun se on mahdollista. Esittele jäsenfunktiot vakioina aina, kun niiden tarkoituksena ei ole muuttaa oliota. Tällöin kääntäjä auttaa virheiden löytämisessä, eikä virheitä tarvitse etsiä itse.

On hyvä ohjelmointitapa esitellä niin monta metodia vakiona kuin mahdollista. Näin kääntäjä löytää hyvin virheet eikä virheitä tarvitse metsästää ajamalla ohjelmaa.

Käyttöliittymä (rajapinta, esittely) ja toteutus (määrittely)

Kuten olet oppinut, ovat asiakkaat sellaisia ohjelman osia, jotka luovat ja käyttävät luokan olioita. Voit ajatella luokan käyttöliittymän - luokan esittelyn - olevan ikään kuin sopimus noiden asiakkaiden kanssa. Sopimus kertoo, mitä tietoa on saatavilla ja kuinka luokka käyttäytyy.

Esimerkiksi Cat-luokan kohdalla solmit sopimuksen, että jokaisella Cat-oliolla on jäsenmuuttuja itsAge, joka voidaan alustaa muodostimessa SetAge()-käyttöfunktion avulla ja jonka arvo voidaan lukea GetAge()-funktiolla. Samalla lupaat, että jokainen Cat-olio osaa naukua funktiolla Meow().

Jos teet GetAge()-funktiosta vakiofunktion (kuten pitäisi tehdä), sopimus lupaa vielä, että GetAge() ei voi mitenkään muokata Cat-oliota.

Miksi käyttää kääntäjää virheiden hakemisessa?
Ohjelmoinnin karu totuus on se, ettei kukaan kirjoita täysin virheetöntä koodia. Ammattilaisen erottaa harrastelijasta - ei koodin virheettömyys - vaan se, että virheet löydetään ennen tuotteen jakelua, ei sen jälkeen.
Kääntämisenaikaiset virheet ovat paljon parempia kuin ajonaikaiset virheet. Tämä johtuu siitä, että kääntämisenaikaiset virheet voidaan löytää paljon luotettavammin. Ohjelmaa on mahdollista ajaa useita kertoja ilman, että joudutaan tutkimaan jokainen koodipaikka. Sen sijaan ajonaikaisia virheitä on vaikea löytää. Kääntämisenaikaiset virheet löydetään kääntämisen yhteydessä, jolloin ne voidaan tunnistaa ja korjata. Tärkeimpiä ohjelman laatutavoitteita on taata, ettei ohjelmassa ole ajonaikaisia virheitä. Kääntämisen käyttäminen tähän on eräs yritys-ja-erehdys -menettelytapa, jolla pyritään löytämään virheet varhaisessa vaiheessa.

Minne luokan esittelyt ja metodien määrittely sijoitetaan

Jokaisella luokassa esitellyllä funktiolla tulee olla määrittelynsä. Määrittelyä kutsutaan myös funktion toteutukseksi. Kuten muillakin funktioilla, myös luokan metodeilla tulee määrittelyn sisältää funktion otsikko ja runko.

Määrittelyn tulee olla tiedostossa, jonka kääntäjä löytää. Useimmat C++ -kääntäjät odottavat, että tiedoston tunnus on .C tai .CPP. Tässä kirjassa käytetään tunnusta .CPP, mutta tarkista oman kääntäjäsi toivoma tunnus.

Huom! Monet kääntäjät olettavat, että C-loppuiset tiedostot ovat C-ohjelmia ja CPP-loppuiset taas C++ -ohjelmia. Voit käyttää jompaakumpaa tunnusta, mutta CPP-tunnuksen käyttäminen vähentää sekaannuksia.

Sinun on lisättävä nuo .CPP-tiedostot projektiisi tai makefile-tiedostoosi. Lisäämistapa riippuu kääntäjästäsi. Jos käytät IDE-työkalua, käytä sellaista komentoa kuin "add files to project". Kaikki .CPP-tiedostot tulee lisätä projektiin, jotta se voidaan kääntää ja linkittää lopulliseksi suoritettavaksi tiedostoksi.

Sijoita luokan esittelyt otsikkotiedostoihin

Vaikka voitkin sijoittaa esittelyt samaan lähdekooditiedostoon, se ei ole hyvä ohjelmointimenettelytapa. Useimmat ohjelmoijat sijoittavat esittelyn niin sanottuun otsikkotiedostoon (header file) käyttäen samaa nimeä, mutta tunnusta .H, .HP tai .HPP. Tämä kirja käyttää tunnusta .HPP, mutta tarkista kääntäjäsi oletusarvo.

Voit laittaa esimerkiksi Cat-luokan esittelyn tiedostoon nimeltä CAT.HPP ja luokan metodien määrittelyt tiedostoon nimeltä CAT.CPP. Sitten yhdistän otsikkotiedoston CPP-tiedostoon #include-säännöllä:

```
#include Cat.hpp
```

Rivi kertoo kääntäjälle, että sen tulee lukea tiedosto CAT.HPP kooditiedostoon aivan kuin olisin kirjoittanut sisällön sinne itse. Miksi sitten kannattaa erottaa tiedot eri tiedostoihin? Useimmin eivät luokan asiakkaat välitä toteutuksen yksityiskohdista. Otsikkotiedoston lukeminen kertoo niille kaiken tarvittavan ja ne voivat ohittaa toteutustiedostot.

Huom! Luokan esittely kertoo kääntäjälle, mikä luokka on; millaista tietoa siihen voidaan tallentaa ja mitä funktioita se sisältää. Luokan esittelyä kutsutaan liittymäksi, koska se kertoo, kuinka luokan kanssa ollaan vuorovaikutuksessa. Tuo käyttöliittymä tallennetaan yleensä .HPP-tiedostoon eli otsikkotiedostoon.

Funktion määrittely kertoo kääntäjälle, kuinka funktio toimii. Määrittelyä kutsutaan luokan metodin toteutukseksi ja se pidetään .CPP-tiedostossa. Vain luokan kirjoittaja tuntee toteutuksen yksityiskohdat. Luokan asiakkaat - ne ohjelman osat, jotka käyttävät luokkaa - eivät välitä tietää (eikä niiden tarvitse tietää), kuinka funktiot on toteutettu.

Inline-toteutus

Kääntäjää voidaan pyytää tekemään tavallisesta funktiosta inline-funktio ja sama voidaan toteuttaa myös metodien kohdalla. Esimerkiksi funktion GetWeight() inline-toteutus olisi seuraavanlainen:

```
inline int Cat::GetWeight()
{
  return itsWeight; //palauttaa Weight-arvon
}
```

Voit sijoittaa myöskin funktion määrittelyn luokan esittelyyn, mikä tekee funktiosta automaattisesti inline-funktion:

```
class Cat
{
public:
  int GetWeight() { return itsWeight; }  //inline
  void SetWeight(int aWeight);
};
```

Huomaa GetWeight()-funktion määrittelyn syntaksi. Inline-funktion runko alkaa heti luokan metodin esittelyn jälkeen eikä aaltosulkujen jälkeen ole

puolipistettä. Määrittely alkaa aloittavalla aaltosululla kuten normaalisti ja päättyy lopettavaan aaltosulkuun. Välilyönneillä ei ole merkitystä ja esittely olisi voitu kirjoittaa myös näin:

```
class Cat
{
public:
  int GetWeight()
  {
    return itsWeight;
  } //inline
  void SetWeight(int aWeight);
};
```

Listauksissa 7.1 ja 7.2 luodaan Cat-luokka uudelleen, mutta esittely sijoitetaan tiedostoon CAT.HPP ja funktioiden toteutus tiedostoon CAT.CPP. Listauksessa 7.1 muutetaan sekä käsittelyfunktiot että Meow() inline-tyyppisiksi.

Listaus 7.1. Cat-luokan esittely tiedostossa CAT.HPP.

```
1:  #include <iostream.h>
2:  class Cat
3:  {
4:  public:
5:    Cat (int initialAge);
6:    ~Cat();
7:    int GetAge() { return itsAge;}            // inline!
8:    void SetAge (int age) { itsAge = age;}    // inline!
9:    void Meow() { cout << "Meow.\n";}           // inline!
10:   private:
11:   int itsAge;
12: };
```

Listaus 7.2. Cat-luokan toteutus tiedostossa CAT.CPP.

```
1:  // inline
2:  // ja include
3:
4:  include "cat.hpp"  // Cat-esittely mukaan!
5:
6:
7:  Cat::Cat(int initialAge)  // Muodostin
8:  {
9:    itsAge = initialAge;
10: }
11:
12: Cat::~Cat()
13: {
14: }
15:
16: // Luodaan Cat, asetetaan Age
17: // ja käytetään Meow-metodia
18: int main()
19: {
20:   Cat Frisky(5);
```

```
21:    Frisky.Meow();
22:    cout << "Frisky is a cat who is ";
23:    cout << Frisky.GetAge() << "years old. \n";
24:    Frisky.Meow();
25:    Frisky.SetAge(7);
26:    cout << "Now Frisky is ";
27:    cout << Frisky.GetAge() << "years old.\n";
28:    return 0;
29: }
```

Tulostus

```
Meow.
Frisky is a cat who is 5 years old.
Meow.
Now Frisky is 7 years old.
```

Analyysi

Listauksessa 7.1 esitellään rivillä 7 funktio GetAge() ja sen inline-toteutus on mukana. Riveillä 8 ja 9 on lisää inline-funktioita, mutta näiden funktioiden toimintaa ei ole muutettu aiemmista ulkoisista toteutuksista.

Listauksen 7.2 rivillä 4 on lause #include "cat.hpp", joka tuo koodiin CAT.HPP-listauksen. Samalla lailla sisällytetään tiedostoon myös IOSTREAM.H, joka antaa käyttöön cout-olion.

Luokat, joiden jäsentietoina on muita luokkia

On aivan normaalia luoda monimutkaisia luokkia, joiden esittelyihin sijoitetaan yksinkertaisempia luokkia. Voimme esitellä esimerkiksi Wheel-luokan, Motor-luokan, Transmission-luokan jne (pyörä, moottori, voimansiirto) ja yhdistää ne Car-luokkaan. Tällöin syntyy sisältymissuhde: autossa on moottori, pyörät ja voimansiirto.

Ajattele toista esimerkkiä. Suorakaide koostuu viivoista. Viiva määritellään kahdella pisteellä. Piste määritellään koordinaatein x ja y. Listaus 7.3 sisältää kokonaisen Rectangle-luokan esittelyn sellaisena kuin se esiintyisi tiedostossa RECTANGLE.HPP. Koska suorakaide määritellään neljällä suoralla viivalla, jotka yhdistävät neljä pistettä, jotka pisteet esitetään koordinaatteina, esitellään ensin Point-luokka, joka sisältää kunkin pisteen x- ja y-koordinaatit. Listaus 7.4 sisältää molempien luokkien esittelyt.

Listaus 7.3. Kokonaisen luokan esittely.

```
1:  // Aloitetaan Rect.hpp
2:  #include <iostream.h>
3:  class Point     // holds x,y coordinates
4:  {
```

```
5:  // ei muodostinta, käytetään oletusmuodostinta
6:  public:
7:    void SetX(int x) { itsX = x; }
8:    void SetY(int y) { itsY = y; }
9:    int GetX()const { return itsX;}
10:   int GetY()const { return itsY;}
11: private:
12:   int itsX;
13:   int itsY;
14: };    // end of Point class declaration
15:
16:
17: class  Rectangle
18: {
19: public:
20:   Rectangle (int top, int left, int bottom, int right);
21:   ~Rectangle () {}
22:
23:   int GetTop() const { return itsTop; }
24:   int GetLeft() const { return itsLeft; }
25:   int GetBottom() const { return itsBottom; }
26:   int GetRight() const { return itsRight; }
27:
28:   Point  GetUpperLeft() const { return itsUpperLeft; }
29:   Point  GetLowerLeft() const { return itsLowerLeft; }
30:   Point  GetUpperRight() const { return itsUpperRight; }
31:   Point  GetLowerRight() const { return itsLowerRight; }
32:
33:   void SetUpperLeft(Point Location)  {itsUpperLeft = Location;}
34:   void SetLowerLeft(Point Location)  {itsLowerLeft = Location;}
35:   void SetUpperRight(Point Location)  {itsUpperRight = Location;}
36:   void SetLowerRight(Point Location)  {itsLowerRight = Location;}
37:
38:   void SetTop(int top) { itsTop = top; }
39:   void SetLeft (int left) { itsLeft = left; }
40:   void SetBottom (int bottom) { itsBottom = bottom; }
41:   void SetRight (int right) { itsRight = right; }
42:
43:   int GetArea() const;
44:
45: private:
46:   Point  itsUpperLeft;
47:   Point  itsUpperRight;
48:   Point  itsLowerLeft;
49:   Point  itsLowerRight;
50:   int    itsTop;
51:   int    itsLeft;
52:   int    itsBottom;
53:   int    itsRight;
54: };
55: // LOPPU Rect.hpp
```

Listaus 7.4. RECT.CPP.

```
1:  // Aloitetaan rect.cpp
2:  #include "rect.hpp"
3:  Rectangle::Rectangle(int top, int left, int bottom, int right)
4:  {
5:    itsTop = top;
6:    itsLeft = left;
7:    itsBottom = bottom;
```

```
8:    itsRight = right;
9:
10:   itsUpperLeft.SetX(left);
11:   itsUpperLeft.SetY(top);
12:
13:   itsUpperRight.SetX(right);
14:   itsUpperRight.SetY(top);
15:
16:   itsLowerLeft.SetX(left);
17:   itsLowerLeft.SetY(bottom);
18:
19:   itsLowerRight.SetX(right);
20:   itsLowerRight.SetY(bottom);
21: }
22:
23:
24: // lasketaan suorakaiteen ala hakemalla kulmat,
25: // lasketaan leveys ja korkeus ja kerrotaan
26: int Rectangle::GetArea() const
27: {
28:   int Width = itsRight - itsLeft;
29:   int Height = itsTop - itsBottom;
30:   return(Width * Height);
31: }
32:
33: int main()
34: {
35:   // alustetaan paikallinen Rectangle-muuttuja
36:   Rectangle MyRectangle(100, 20, 50, 80);
37:
38:   int Area = MyRectangle.GetArea();
39:
40:   cout << "Area: " << Area << "\n";
41:   cout << Upper  Left  X Coordinate: ";
42:   cout << MyRectangle.GetUpperLeft().GetX();
43:   return 0;
44: }
```

Tulostus

```
Area: 3000
Upper Left X Coordinate: 20
```

Analyysi

Listauksen 7.3 rivit 3-14 esittelevät luokan Point, joka sisältää pisteen x- ja y-koordinaatit. Tämä ohjelma ei hyödynnä Points-olioita paljoakaan. Kuitenkin muut piirtomenetelmät vaativat Points-olioita.

Point-luokassa esitellään kaksi jäsenmuuttujaa, itsX ja itsY, riveillä 12 ja 13. Nämä muuttujat tallentavat koordinaattiarvot. Kun x-koordinaatti kasvaa, siirrytään koordinaatistossa oikealle. Kun y-koordinaatti kasvaa, siirrytään ylöspäin. Muut graafit voivat käyttää muunlaista systeemiä. Esimerkiksi joissakin ikkunointiohjelmissa kasvatetaan y-koordinaattia siirryttäessä ikkunassa alaspäin.

Point-luokka käyttää inline-tyyppisiä funktioita hakemaan tai asettamaan x- ja y-pisteitä. Points käyttää oletusmuodostinta ja tuhoajaa ja siksi niiden koordinaatit on asetettava ulkoisesti.

Rivi 17 aloittaa Rectangle-luokan esittelyn. Se koostuu neljästä pisteestä, jotka edustavat suorakaiteen nurkkia.

Rectangle-luokan muodostin (rivi 20) käyttää neljää kokonaislukua, top, left, bottom ja right. Nuo parametrit kopioidaan neljään jäsenmuuttujaan, jolloin toteutetaan neljä pistettä.

Rectangle-luokassa on myös funktio nimeltä GetArea(), joka esitellään rivillä 43. Tämä funktio laskee alan riveillä 27-29 laskemalla ensin suorakaiteen leveyden ja korkeuden ja kertomalla nuo luvut keskenään.

Vasemman ylänurkan x-koordinaatti saadaan UpperLeft-pisteen X-arvosta. Koska GetUpperLeft() on Rectangle-luokan metodi, se voi käsitellä suoraan Rectangle-luokan yksityistä tietoa mukaan lukien itsUpperLeft. Koska itsUpperLeft on Point ja Pointin itsX on yksityinen arvo, ei GetUpperLeft() voi käsitellä tätä tietoa suoraan. Pikemminkin sen täytyy käyttää julkista käsittelyfunktiota getX() saadakseen tuon arvon.

Listauksen 7.4 rivi 32 on todellisen ohjelman rungon alku. Vasta rivillä 35 varataan muistia. Sitä ennen kerrotaan kääntäjälle, kuinka Point ja Rectangle toteutetaan.

Rivillä 36 määritellään Rectangle viemällä sille top-, left-, bottom- ja right-arvot.

Rivillä 38 määritellään paikallinen int-muuttuja, Area, johon tallennetaan suorakaiteen ala. Area alustetaan Rectangle-luokan GetArea()-funktion palauttamalla arvolla.

Rectangle-luokan asiakas voisi luoda Rectangle-olion ja saada sen alan etsimättä koskaan GetArea()-funktion toteutusta.

Tutkimalla otsikkotiedostoa ohjelmoija näkee, että GetArea() palauttaa kokonaisluvun, mutta Rectangle-luokan asiakkaan ei tarvitse huolehtia siitä, kuinka GetArea() toimii. Itse asiassa Rectangle-luokan kirjoittaja voisi muuttaa GetArea()-funktiota vaikuttamatta muutoksilla ohjelmiin, jotka käyttävät Rectangle-luokkaa.

Yhteenveto

Tässä luvussa opit lisää luokkien luomisesta.

Opit luomaan vakioita jäsenfunktioita ja tuntemaan luokan metodien esittelyn ja toteutuksen erot.

On hyvä ohjelmointitapa eristää luokan esittely omaan otsikkotiedostoonsa. Yleensä tuon tiedoston tunnuksena on .HPP. Luokan metodien toteutukset koodataan tiedostoon, jonka tunnus on .CPP.

Kysymyksiä ja Vastauksia

K
Jos const-tyyppisen funktion käyttäminen luokan muuttamiseen aiheuttaa kääntäjän virheen, miksi en voisi jättää pois tuota const-sanaa virheiden välttämiseksi?
V
Jos jäsenfunktion ei pitäisi muuttaa luokkaa, voidaan const-avainsanalla estää typerien virheiden tekeminen. Jos vaikkapa GetAge()-funktion ei tulisi muuttaa luokan tietoja, mutta toteutuksessasi on kuitenkin seuraava lause:

```
if (itsAge = 100) cout << "Hey! You're 100 years old\n";
```

Aiheuttaisi const-avainsana virheen. Tarkoituksesi oli tietenkin testata, onko itsAge yhtä suuri kuin 100, mutta sen sijaan aiheuttaa kirjoitusvirhe sijoitusoperaation ja sijoittaminen sotii const-avainsanaa vastaan. Nyt kääntäjä kuitenkin havaitsee virheen, jos olet käyttänyt const-avainsanaa.

Juuri tuollaiset virheet ovat vaikeita havaita, koska silmä näkee usein sen, mitä odotetaan nähtävän. Lisäksi ohjelmaa voidaan ajaa, vaikka itsAge on saanut erikoisen arvon, mikä taas voi aiheuttaa virheitä myöhemmin.

K
Onko C++ -ohjelmassa koskaan syytä käyttää tietue-rakennetta?
V
Monet C++ -ohjelmoijat varaavat struct-avainsanan sellaisten luokkien käyttöön, joissa ei ole funktioita. Tämä on jäänne vanhoista C-tietueista, joissa ei voitu käyttää funktioita. Minusta kyseessä on huono ohjelmointitapa. Nykyiset funktiottomat tietueet saattavat tarvita funktioita tulevaisuudessa. Tällöin ohjelmoijan on muutettava tyyppi class-tyypiksi tai rikottava sääntöjään vastaan ja päädyttävä tietueisiin, joissa on metodeita.