Osa V

Periytyminen ja monimuotoisuus

Oppitunnit

Osa
V

16. oppitunti

Periyttäminen

Ihmiselle on luonteenomaista etsiä, tunnistaa ja luoda yhteyksiä eri käsitteiden välille. Muodostamme hierarkioita, matriiseja, verkkoja ja muita vuorovaikutussuhteita selittääksemme ja ymmärtääksemme eri asioiden keskinäisiä suhteita. C++ yrittää siepata tuon ajattelutavan hierarkioihin. Tässä luvussa käsitellään seuraavia asioita:

- Mitä on periytyminen
- Kuinka luokasta johdetaan uusi luokka
- Mitä on suojattu käsittely ja kuinka sitä käytetään

Mitä on periytyminen?

Mikä on koira? Kun katsot lemmikkiäsi, mitä näet? Biologi näkee yhdessä toimivien elinten verkoston. Fyysikko näkee atomeita ja voimia työssä. Kehitysoppitieteilijä näkee tietyn Canis familiarus -lajin edustajan ja äitini taas koiran karvat ja kuonon.

Kehitysoppitieteilijän perspektiivi kiinnostaa meitä eniten (anteeksi, äiti). Hän jakaa elävien olentojen maailman kuningaskuntaan, phylum, luokkaan, järjestykseen, perheeseen, heimoon ja lajiin.

Tällaisessa hierarkiassa nähdään on-suhde. Tällaisia suhteita on kaikkialla: Toyota on auto, joka on taas ajoneuvo. Kiisseli on jonkinlainen jälkiruoka, joka on taas ruoka.

Autot ja bussit ovat molemmat ajoneuvoja, mutta niillä on omat erikoispiirteensä.

Periytyminen ja johtaminen

Käsite koira perii kaikki eläimen piirteet. Koska kyseessä on eläin, me tiedämme, että se liikkuu ja hengittää, koska niin eläimet yleensä tekevät. Käsite koira lisää ajatteluun vielä haukkumisen, hännän heilutuksen, jne. Koiran erikoispiirteet kuuluvat koiralle, kun taas eläimen piirteet kuuluvat kaikille eläimille.

Voimme edelleen jakaa koirat metsästyskoiriin ja terriereihin ja terrierit voimme taas jakaa Yorkshiren terriereihin, Dandie Dinmontint terriereihin jne.

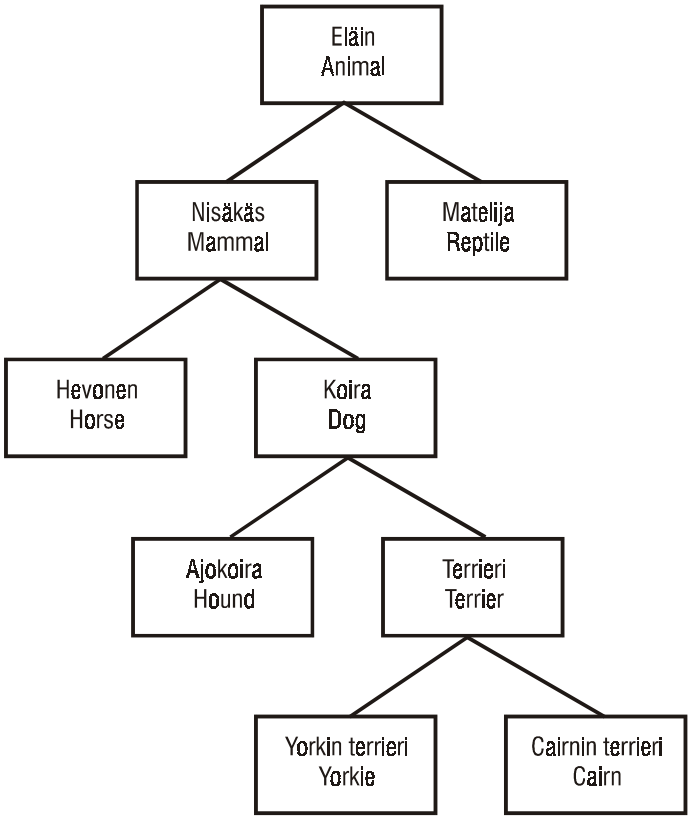
Yorkshire-terrieri kuuluu terriereihin ja siksi se kuuluu myös koiriin. Samalla se kuuluu eläimiin ja sitä kautta eläviin olentoihin. Kuva 16.1 kuvaa tätä hierarkiaa.

C++ yrittää esittää noita suhteita määrittelemällä luokkia, jotka johdetaan toisista luokista. Johtaminen on tapa ilmaista on-suhdetta. Johdamme uuden luokan, Dog, luokasta Mammal. Meidän ei tarvitse enää määrittää, että koira liikkuu, koska se ominaisuus periytyy Mammal-luokasta.

Luokka, joka lisää uusia ominaisuuksia olemassaolevaan luokkaan, johdetaan tuosta alkuperäisestä luokasta. Alkuperäisen luokan sanotaan olevan tuon uuden luokan perusluokka.

Jos Dog-luokka johdetaan Mammal-luokasta, on Mammal Dog-luokan perusluokka. Johdetut luokat ovat perusluokkiensa perusjoukkoja. Koira lisää omia ominaisuuksiaan eläin-käsitteeseen ja samalla lailla Dog-luokka lisää joitakin metodeita ja tietoja Mammal-luokkaan.

Yleensä perusluokalla on useampia kuin yksi johdettu luokka. Samalla lailla kuin kissat, koirat ja hevoset ovat eläimiä, vastaavat luokatkin johdetaan Mammal-luokasta.



Kuva 16.1. Eläinhierarkia.

Eläinten valtakunta

Tutkailemme vielä periytymistä ja johtamista eläinnäkökulmasta lähtien. Olettakaamme, että sinun tulisi suunnitella lasten peli - maatilan simulointi.

Kaikki maatilan eläimet tulee kehittää: hevoset, lehmät, koirat, kissat, lampaat, jne. Luot vastaavat luokat ja niihin metodit, jotta ne voisivat toimia lasten odottamalla tavalla. Tällä erää demoat kuitenkin noita metodeja pelkästään tulostamalla viestejä.

Funktion demoaminen kuvaa funktion kutsua. Funktion voi laatia myöhemmin.

Laita itse lisää koodia funktioihin saadaksesi eläimet toimimaan paremmin. Sitten, kun sinulla on aikaa siihen…

Johtamisen syntaksi

Kun esittelet luokan, johtamisen lähteenä oleva luokka kerrotaan kirjoittamalla kaksoispiste luokan nimen jälkeen, johtamisen tyyppi

(julkinen tai muu) ja lopuksi tulee johtamisen lähteenä olevan luokan nimi. Seuraavassa on esimerkki:

```
class Dog : public Mammal
```

Johtamisen tyyppiä käsitellään myöhemmin tässä luvussa. Toistaiseksi tyyppinä on public, julkinen. Perusluokka tulee olla määritettynä aiemmin, tai kääntäjä antaa virheilmoituksen. Listaus 16.1 havainnollistaa Mammal-luokasta johdettua Dog-luokan esittelyä.

Listaus 16.1. Yksinkertainen periytyminen.

```
1:  //Listaus 16.1 Suppea periyttäminen
2:
3:  #include <iostream.h>
4:  enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
5:
6:  class Mammal
7:  {
8:  public:
9:    // muodostimet
10:   Mammal();
11:   ~Mammal();
12:
13:   //käsittelijät
14:   int GetAge()const;
15:   void SetAge(int);
16:   int GetWeight() const;
17:   void SetWeight();
18:
19:   //muut metodit
20:   void Speak();
21:   void Sleep();
22:
23:
24: protected:
25:   int itsAge;
26:   int itsWeight;
27: };
28:
29: class Dog : public Mammal
30: {
31: public:
32:
33:   // muodostimet
34:   Dog();
35:   ~Dog();
36:
37:   // käsittelijät
38:   BREED GetBreed() const;
39:   void SetBreed(BREED);
40:
41:   // muuta metodit
42:   // WagTail();
43:   // BegForFood();
44:
45: protected:
46:   BREED itsBreed;
47: };
```

Tulostus
Tämä esimerkki ei tulosta mitään, koska siinä on vain luokkaesittelyjä ilman niiden toteutuksia.

Analyysi
Riveillä 6-27 esitellään Mammal-luokka. Tässä esimerkissä ei Mammal-luokkaa johdeta muista luokista. Todellisessa elämässä eläimet ovat eläviä olentoja. C++ -ohjelmassa voimme esittää vain murto-osan siitä informaatiosta, mikä meillä on annetusta kohteesta. Todellisuus on aivan liian monimutkainen esiteltäväksi kokonaisuudessaan, joten jokainen c++ -hierarkia on vain käytettävissä olevan tiedon esitys. Hyvän suunnittelun pohjana on esittää välttämättömät alueet siten, että ne vastaavat todellisuutta järkevän uskollisesti.

Hierarkian tulee alkaa jostakin; tämä ohjelma alkaa Mammal-luokasta. Tästä johtuen jotkut jäsenmuuttujat, jotka saattaisivat kuulua ylempään perusluokkaan, ovat edustettuina täällä. Tietenkin kaikilla eläimillä on esimerkiksi ikä ja paino, joten Animal-luokasta johdettu Mammal perii nuo ominaisuudet. Nyt ominaisuudet ovat Mammal-luokassa esiteltyjä.

Pitääksemme ohjelman tarpeeksi yksinkertaisena ja hallittavana on Mammal-luokkaan sijoitettu vain kuusi metodia: neljä käsittelymetodia sekä Speak() ja Sleep().

Dog-luokka perii kaiken Mammal-luokalta, kuten rivi 29 osoittaa. Jokainen Dog-olio omistaa kolme jäsenmuuttujaa: itsAge, itsWeight ja itsBreed. Huomaa, että Dog ei sisällä jäsenmuuttujia itsAge ja itsWeight, vaan ne periytyvät Mammal-luokasta. Samoin periytyvät metodit lukuun ottamatta kopiomuodostinta, muodostimia ja tuhoajafunktiota.

Yksityinen (private) vastaan suojattu (protected)

Olet saattanut huomata, että riveillä 24-25 on käytetty uutta avainsanaa, protected. Aiemmin on luokan tieto esitelty private-tyyppisenä. Private-tyyppiset jäsenet eivät ole kuitenkaan käsiteltävissä johdetuista luokista. Olisimme voineet esitellä jäsenmuuttujat itsAge ja itsWeight public-tyyppisinä, mutta se ei ole toivottavaa. Emme halua muiden luokkien käsittelevän näitä tietojäseniä suoraan.

Haluamme, että nyt saamme aikaan sen, että "nuo tiedot näkyvät tälle luokalle ja tästä luokasta johdetuille luokille". Protected-tyyppi sallii sen. Protected-tyyppiset tietojäsenet ja funktiot näkyvät täysin johdetuille luokille, mutta ovat muutoin private-tyyppisiä.

Käyttömääritteitä on kolmea eri tyyppiä: public, protected ja private. Jos funktio käsittelee luokan olioita, se voi käsitellä sen kaikkia public-tyyppisiä tietoja ja funktioita. Jäsenfunktiot voivat vuorostaan käsitellä kaikkia oman luokkansa private-tyyppisiä tietojäseniä ja funktioita sekä kaikkia perusluokkiensa protected-tyyppisiä tietojäseniä ja funktioita.

Siksi funktio Dog::WagTail() voi käsitellä private-tyyppistä itsBreed-jäsentä sekä Mammal-luokan protected-tyyppistä tietoa.

Vaikka muita luokkia olisi kerrostettuina Mammal-luokan ja Dog-luokan väliin (esimerkiksi DomesticAnimals), Dog-luokka voi silti käsitellä Mammal-luokan protected-tyyppisiä jäseniä, olettaen että kaikki muut luokat käyttävät public-tyyppistä periytymistä.

Listaus 16.2 esittelee Dog-tyyppisten olioiden luomista ja jäsenten käyttöä.

Listaus 16.2. Johdetun olion käyttö.

```
1:  //Listaus 16.2 Johdetun olion käyttö
2:
3:  #include <iostream.h>
4:
5:  enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
6:
7:  class Mammal
8:
9:  {
10: public:
11:   // muodostimet
12:   Mammal():itsAge(2), itsWeight(5){}
13:   ~Mammal(){}
14:
15:   //käsittelijät
16:   int GetAge()const { return itsAge; }
17:   void SetAge(int age) { itsAge = age; }
18:   int GetWeight() const { return itsWeight; }
19:   void SetWeight(int weight) { itsWeight = weight; }
20:
21:   //muut metodit
22:   void Speak()const { cout << "Mammal sound!\n"; }
23:   void Sleep()const { cout << "shhh. I'm sleeping.\n"; }
24:
25:
26: protected:
27:   int itsAge;
28:   int itsWeight;
29: };
30:
31: class Dog : public Mammal
32: {
33: public:
34:
35:   // muodostimet
36:   Dog():itsBreed(YORKIE){}
37:   ~Dog(){}
38:
39:   // käsittelijät
40:   BREED GetBreed() const { return itsBreed; }
```

```
41:    void SetBreed(BREED breed) { itsBreed = breed; }
42:
43:    // muut metodit
44:    void WagTail() { cout << "Tail wagging...\n"; }
45:    void BegForFood() { cout << "Begging for food...\n"; }
46:
47:  private:
48:    BREED itsBreed;
49:  };
50:
51:  int main()
52:  {
53:    Dog fido;
54:    fido.Speak();
55:    fido.WagTail();
56:    cout << "Fido is " << fido.GetAge() << " years old\n";
57:    return 0;
58:  }
```

Tulostus

```
Mammal sound!
Tail wagging…
Fido is 2 years old
```

Analyysi

Riveillä 7-29 esitellään Mammal-luokka (kaikki sen funktiot ovat tilan säästämiseksi inline-tyyppisiä). Riveillä 31-49 esitellään Dog-luokka johdettuna Mammal-luokasta. Niinpä nyt kaikilla Dog-olioilla on ominaisuudet ikä, paino ja ravinto.

Rivillä 53 esitellään Dog-olio, Fido. Se perii kaikki ominaisuudet Mammal-luokalta sekä myös Dog-luokalta. Siten Fido osaa käyttää toimintoa WagTail() ja lisäksi toimintoja Speak() ja Sleep().

Muodostimet ja tuhoajafunktiot

Dog-oliot ovat Mammal-olioita. Tämä onkin on-suhteen ydin. Kun Fido luodaan, sen perusmuodostinta kutsutaan ensin, jolloin luodaan Mammal. Sitten kutsutaan Dog-muodostinta, joka täydentää Dog-olion rakenteen. Koska Fidolle ei annettu mitään parametreja, molemmissa tapauksissa kutsutaan oletusmuodostinta. Fido ei ole olemassa ennen kuin se on kokonaan muodostettu, eli sekä Mammal-osuus että Dog-osuus on muodostettava ensin. Kumpaakin muodostinta täytyy siis kutsua.

Kun Fido tuhotaan, kutsutaan ensin Fidon tuhoajafunktiota ja sen jälkeen tuhotaan Fidon Mammal-osuus. Kumpikin tuhoajafunktio saa luvan perata oma osansa Fidosta. Muista käyttää tuhoajafunktiota Dog-olioille! Listaus 16.3 esittelee tätä.

Listaus 16.3. Muodostimien ja tuhoajafunktioiden kutsuminen.

```
1:  //Listaus 16.3 Muodostimien ja tuhoajien kutsuminen.
2:
3:  #include <iostream.h>
4:  enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
5:
6:  class Mammal
7:  {
8:  public:
9:    // muodostimet
10:   Mammal();
11:   Mammal();
12:
13:   //käsittelijät
14:   int GetAge() const { return itsAge; }
15:   void SetAge(int age) { itsAge = age; }
16:   int GetWeight() const { return itsWeight; }
17:   void SetWeight(int weight) { itsWeight = weight; }
18:
19:   //muut metodit
20:   void Speak() const { cout << "Mammal sound!\n"; }
21:   void Sleep() const { cout << "shhh. I'm sleeping.\n"; }
22:
23:
24: protected:
25:   int itsAge;
26:   int itsWeight;
27: };
28:
29: class Dog : public Mammal
30: {
31: public:
32:
33:   // muodostimet
34:   Dog();
35:   ~Dog();
36:
37:   // käsittelijät
38:   BREED GetBreed() const { return itsBreed; }
39:   void SetBreed(BREED breed) { itsBreed = breed; }
40:
41:   // muut metodit
42:   void WagTail() { cout << "Tail wagging...\n"; }
43:   void BegForFood() { cout << "Begging for food...\n"; }
44:
45: private:
46:   BREED itsBreed;
47: };
48:
49: Mammal::Mammal():
50: itsAge(1),
51: itsWeight(5)
52: {
53:   cout << "Mammal constructor...\n";
54: }
55:
56: Mammal::~Mammal()
57: {
58:   cout << "Mammal destructor...\n";
59: }
60:
```

```
61: Dog::Dog():
62: itsBreed(YORKIE)
63: {
64:   cout << "Dog constructor...\n";
65: }
66:
67: Dog::~Dog()
68: {
69:   cout << "Dog destructor...\n";
70: }
71:
72: int main()
73: {
74:   Dog fido;
75:   fido.Speak();
76:   fido.WagTail();
77:   cout << "Fido is " << fido.GetAge() << " years old\n";
78:   return 0;
79: }
```

Tulostus

```
Mammal constructor…
Dog constructor…
mammal sound!
Tail wagging…
Fido is 1 years old
Dog destructor…
Mammal destructor…
```

Analyysi

Listaus 16.3 on muutoin samanlainen kuin 16.2, paitsi että nyt tulostetaan viesti ruudulle kutsuttaessa muodostimia ja tuhojafunktioita. Mammal-luokan muodostinta kutsutaan ensin ja sitten Dog-luokan. Tämän jälkeen on Dog olemassa ja sen metodeita voidaan käyttää. Kun Fido katoaa näkyvyysalueelta, kutsutaan Dogin tuhoajafunktiota ja sen jälkeen Mammal-luokan tuhoajafunktiota.

Argumenttien vieminen perusmuodostimille

On mahdollista ylikuormittaa Mammal-muodostin ottamaan parametrikseen tietty ikä sekä Dog-muodostin alustamaan rotu. Kuinka ikä- ja painoparametrit saadaan viedyiksi oikealle Mammal-muodostimelle? Entäpä, jos Dog haluaa alustaa painon, mutta Mammal ei?

Perusluokan alustus voidaan toteuttaa luokan alustamisen yhteydessä kirjoittamalla luokan nimi sekä perusluokan odottamat parametrit. Listaus 16.4 havainnollistaa tätä.

Listaus 16.4. Muodostimien ylikuormittaminen johdetuissa luokissa.

```
1:  //Listaus 16.4 Muodostimien ylikuormitus
2:
3:  #include <iostream.h>
4:  enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
```

```
5:
6:  class Mammal
7:  {
8:  public:
9:    // muodostimet
10:   Mammal();
11:   Mammal(int age);
12:   ~Mammal();
13:
14:   //käsittelijät
15:   int GetAge() const { return itsAge; }
16:   void SetAge(int age) { itsAge = age; }
17:   int GetWeight() const { return itsWeight; }
18:   void SetWeight(int weight) { itsWeight = weight; }
19:
20:   //muut metodit
21:   void Speak() const { cout << "Mammal sound!\n"; }
22:   void Sleep() const { cout << "shhh. I'm sleeping.\n"; }
23:
24:
25: protected:
26:   int itsAge;
27:   int itsWeight;
28: };
29:
30: class Dog : public Mammal
31: {
32: public:
33:
34:   //muodostimet
35:   Dog();
36:   Dog(int age);
37:   Dog(int age, int weight);
38:   Dog(int age, BREED breed);
39:   Dog(int age, int weight, BREED breed);
40:   ~Dog();
41:
42:   //käsittelijät
43:   BREED GetBreed() const { return itsBreed; }
44:   void SetBreed(BREED breed) { itsBreed = breed; }
45:
46:   //muut metodit
47:   void WagTail() { cout << "Tail wagging...\n"; }
48:   void BegForFood() { cout << "Begging for food...\n"; }
49:
50: private:
51:   BREED itsBreed;
52: };
53:
54: Mammal::Mammal():
55: itsAge(1),
56: itsWeight(5)
57: {
58:   cout << "Mammal constructor...\n";
59: }
60:
61: Mammal::Mammal(int age):
62: itsAge(age),
63: itsWeight(5)
64: {
65:   cout << "Mammal(int) constructor...\n";
```

```
66: }
67:
68: Mammal::~Mammal()
69: {
70:   cout << "Mammal destructor...\n";
71: }
72:
73: Dog::Dog():
74: Mammal(),
75: itsBreed(YORKIE)
76: {
77:   cout << "Dog constructor...\n";
78: }
79:
80: Dog::Dog(int age):
81: Mammal(age),
82: itsBreed(YORKIE)
83: {
84:   cout << "Dog(int) constructor...\n";
85: }
86:
87: Dog::Dog(int age, int weight):
88: Mammal(age),
89: Breed(YORKIE)
90: {
91:   itsWeight = weight;
92:   cout << "Dog(int, int) constructor...\n";
93: }
94:
95: Dog::Dog(int age, int weight, BREED breed):
96: Mammal(age),
97: itsBreed(breed)
98: {
99:   itsWeight = weight;
100:  cout << "Dog(int, int, BREED) constructor...\n";
101: }
102:
103: Dog::Dog(int age, BREED breed):
104: Mammal(age),
105: itsBreed(breed)
106: {
107:  cout << "Dog(int, BREED) constructor...\n";
108: }
109:
110: Dog::~Dog()
111: {
112:  cout << "Dog destructor...\n";
113: }
114: int main()
115: {
116:  Dog fido;
117:  Dog rover(5);
118:  Dog buster(6,8);
119:  Dog yorkie (3,YORKIE);
120:  Dog dobbie (4,20,DOBERMAN);
121:  fido.Speak();
122:  rover.WagTail();
123:  cout << "Yorkie is " << yorkie.GetAge() << " years old\n";
124:  cout << "Dobbie weighs " << dobbie.GetWeight() << " pounds\n";
125:  return 0;
```

```
126: }
```

Huom! Tulostus on numeroitu, jotta analyysissä voidaan viitata rivinumeroihin. Ohjelma ei tulosta rivinumeroita.

Tulostus

```
1:  Mammal constructor…
2:  Dog constructor…
3:  Mammal(int) constructor…
4:  Dog(int) constructor…
5:  Mammal(int) constructor…
6:  Dog(int, int) constructor…
7:  Mammal(int) constructor…
8:  Dog(int, BREED) constructor…
9:  Mammal(int) constructor…
10:  Dog(int, int, BREED) constructor…
11:  Mammal sound!
12:  Tail wagging…
13:  Yorkie is 3 years old
14:  Dobbie weighs 20 pounds
15:  Dog destructor…
16: Mammal destructor…
17:  Dog destructor…
18: Mammal destructor…
19:  Dog destructor…
20: Mammal destructor…
21:  Dog destructor…
22: Mammal destructor…
23:  Dog destructor…
24: Mammal destructor…
```

Analyysi

Mammal-luokan muodostin on ylikuormitettu rivillä 11 ottamaan kokonaislukuparametriksi iän. Rivien 61-66 toteutus alustaa muuttujan itsAge muodostimelle viedyllä arvolla ja muuttujan itsWeight arvolla 5.

Dogin kohdalla on ylikuormitettu viisi muodostinta (rivit 35-39). Ensimmäisenä on oletusmuodostin. Toinen ottaa parametrikseen iän, kuten Mammal-luokan kohdalla. Kolmas muodostin ottaa parametreikseen sekä iän että painon, neljäs ottaa parametreikseen iän ja ravinnon ja viides taas iän, painon ja ravinnon.

Huomaa, että rivin 74 Dogin oletusmuodostin kutsuu Mammal-luokan oletusmuodostinta. Vaikka ei olekaan aivan välttämätöntä tehdä niin, koodi toimii dokumenttina kertoen, että aioit kutsua perusmuodostinta, joka ei ota parametreja. Perusmuodostinta kutsuttaisiin joka tapauksessa, mutta tämä on selvempää.

Dog-muodostimen, joka ottaa kokonaisluvun, toteutus on riveillä 80-85. Sen alustusvaiheessa (rivit 81-82) alustaa Dog perusluokkansa parametrilla ja sen jälkeen alustetaan ravinto.

Toinen Dog-muodostin toteutetaan riveillä 87-93. Se ottaa kaksi parametria. Jälleen kerran se alustaa perusluokkansa kutsumalla sopivaa muodostinta, mutta tällä kertaa se myös sijoittaa painon perusluokkansa itsWeight-muuttujaan. Huomaa, että perusluokan muuttujaan ei voida sijoittaa arvoa alustusvaiheessa. Koska Mammal-luokalla ei ole muodostinta, joka ottaa parametrin, on se tehtävä Dog-muodostimen rungossa.

Käy läpi muut muodostimet voidaksesi ymmärtää niiden toiminnan. Kiinnitä huomio siihen, mitä alustetaan ja mitä taas toteutetaan muodostimen rungossa.

Tulostuksen kaksi ensimmäistä riviä kertovat Fidon luomisesta oletusmuodostinta käyttäen.

Rivit 3 ja 4 edustavat Roverin luomista ja rivit 5 ja 6 Busterin luomista. Huomaa, että kutsuttu Mammal-muodostin ottaa kokonaislukuparametrin, mutta Dog-muodostin taas ottaa kaksi kokonaislukuparametria.

Lopuksi oliot on luotu, niitä on käytetty ja ne poistuvat näkyvyysalueelta. Kun kukin olio tuhotaan, kutsutaan ensin Dog-tuhoajafunktiota ja sen jälkeen Mammal-tuhoajafunktiota; kutsupareja on yhteensä viisi kappaletta.

Funktioiden korvaaminen

Dog-oliolla on pääsy kaikkiin Mammal-luokan jäsenfunktioihin sekä kaikkiin Dog-luokan metodeihin, kuten WagTail(). Dog-luokka voi myöskin korvata perusluokan funktion. Funktion korvaaminen tarkoittaa perusluokan funktion toteutuksen muuttamista johdetussa luokassa. Kun johdetun luokan olio luodaan, kutsutaan oikeata funktiota.

Kun johdettu luokka luo funktion, jolla on sama palautustyyppi ja koko nimi kuin perusluokan jäsenfunktiolla, mutta uusi toteutus, puhutaan metodin korvaamisesta.

Korvatun funktion palautustyypin ja koko nimen tulee olla sama kuin perusluokan funktiolla. Koko nimi on funktion prototyyppi ilman palautustyyppiä; eli se sisältää funktion nimen, parametrit sekä mahdollisesti const-avainsanan.

Funktion koko nimi on sen nimi ja parametrit, mutta se ei siis sisällä palautustyyppiä.

Listaus 16.5 havainnollistaa mitä tapahtuu, kun Dog-luokka korvaa Mammal-luokan Speak()-metodin. Tilan säästämiseksi on käsittelyfunktiot jätetty luokkien ulkopuolelle.

Listaus 16.5. Perusluokan metodin korvaaminen johdetussa luokassa.

```
1:  //Listaus 16.5 Perusluokan metodin korvaaminen
2:
3:  #include <iostream.h>
4:  enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
5:
6:  class Mammal
7:  {
8:  public:
9:    // muodostimet
10:   Mammal() { cout << "Mammal constructor...\n"; }
11:   ~Mammal() { cout << "Mammal destructor...\n"; }
12:
13:   //muut metodit
14:   void Speak()const { cout << "Mammal sound!\n"; }
15:   void Sleep()const { cout << "shhh. I'm sleeping.\n"; }
16:
17:
18: protected:
19:   int itsAge;
20:   int itsWeight;
21: };
22:
23: class Dog : public Mammal
24: {
25: public:
26:
27:   //muodostimet
28:   Dog(){ cout << "Dog constructor...\n"; }
29:   ~Dog(){ cout << "Dog destructor...\n"; }
30:
31:   // muut metodit
32:   void WagTail() { cout << "Tail wagging...\n"; }
33:   void BegForFood() { cout << "Begging for food...\n"; }
34:   void Speak()const { cout << "Woof!\n"; }
35:
36: private:
37:   BREED itsBreed;
38: };
39:
40: int main()
41: {
42:   Mammal bigAnimal;
43:   Dog fido;
44:   bigAnimal.Speak();
45:   fido.Speak();
46:   return 0;
47: }
```

Tulostus

```
Mammal constructor…
Mammal constructor…
Dog constructor…
Mammal sound!
```

```
Woof!
Dog destructor…
Mammal destructor…
Mammal destructor…
```

Analyysi

Rivillä 34 korvaa Dog-luokka Speak()-metodin siten, että Dog-oliot sanovat Woof! Speak()-metodia kutsuttaessa. Rivillä 42 luodaan Mammal-olio, bigAnimal, mikä aiheuttaa ensimmäisen tulostusrivin muodostinta kutsuttaessa. Rivillä 43 luodaan Dog-olio, fido, jolloin syntyy kaksi tulostusriviä. Miksi?

Rivillä 44 kutsuu Mammal-olio Speak()-metodia; sen jälkeen kutsuu Dog-olio omaa Speak()-metodiaan rivillä 45. Tulostus kertoo, että kutsutaan oikeita metodeita. Lopuksi kutsutaan tuhoajafunktioita.

Ylikuormittaminen vastaan korvaaminen

Nämä käsitteet ovat samanlaisia ja tekevät saman asian. Kun ylikuormitat metodin, luot useamman kuin yhden samannimisen metodin, joilla on erilainen koko nimi. Kun korvaat metodin, luot johdettuun luokkaan metodin, jolla on perusluokkansa metodin nimi ja sama koko nimi.

Perusluokan metodin kätkeminen

Edellisessä listauksessa Dog-luokan metodi Speak() kätkee perusluokan metodin. Juuri niin haluttiinkin tehdä, mutta sillä voi olla odottamattomia tuloksia. Jos Mammal-luokalla on metodi Move(), joka on ylikuormitettu ja Dog korvaa tuon metodin, niin Dogin metodi kätkee kaikki tuon nimiset Mammal-metodit.

Jos Mammal ylikuormittaa Move()-metodin kolmeksi metodiksi - yksi ei ota parametreja, yksi ottaa kokonaislukuparametrin ja yksi ottaa kokonaisluvun ja desimaaliluvun parametreikseen - ja Dog korvaa parametrittoman Move()-metodin, ei ole helppoa käyttää noita kahta muuta metodia Dog-oliosta käsin. Listaus 16.6 havainnollistaa tätä ongelmaa.

Listaus 16.6. Metodien kätkeminen.

```
1:  //Listaus 16.6 Metodien kätkeminen
2:
3:  #include <iostream.h>
4:
5:  class Mammal
6:  {
7:  public:
8:    void Move() const { cout << "Mammal move one step\n"; }
9:    void Move(int distance) const
10:     { cout << "Mammal move " << distance <<" _steps.\n"; }
11: protected:
```

```
12:    int itsAge;
13:    int itsWeight;
14: };
15:
16: class Dog : public Mammal
17: {
18: public:
19:    void Move() const { cout << "Dog move 5 steps.\n"; }
20: }; // Saat ehkä varoituksen metodin kätkemisestä!
21:
22: int main()
23: {
24:    Mammal bigAnimal;
25:    Dog fido;
26:    bigAnimal.Move();
27:    bigAnimal.Move(2);
28:    fido.Move();
29:    // fido.Move(10);
30:    return 0;
31: }
```

Tulostus

```
Mammal move one step
Mammal move 2 steps
Dog move 5 steps
```

Analyysi

Kaikki ylimääräiset metodit ja tiedot on poistettu näistä luokista. Riveillä 8-9 esittelee Mammal-luokka ylikuormitetut Move()-metodit. Rivillä 18 korvaa Dog parametrittoman Move()-metodin. Metodia käytetään riveillä 25-27, ja tulostus kertoo vaikutukset.

Rivi 29 on kuitenkin kommentoitu ulos, koska se aiheuttaa kääntäjän virheen. Dog olisi voinut kutsua Move(int)-metodia, jos sillä ei olisi ollut vastaavaa korvattua versiota parametrittomasta Move()-metodista. Nyt Dogin tulisi korvata molemmat Move()-versiot voidaakseen kutsua kumpaakin. Sääntöhän oli se, että, jos kirjoitetaan oma muodostin, ei kääntäjä enää tarjoa oletusmuodostinta.

On yleinen virhe kätkeä perusluokan metodi silloin, kun aiotaan korvata se unohtamalla käyttää const-avainsanaa. const on osa allekirjoitusta ja sen pois jättäminen muuttaa allekirjoitusta ja pikemminkin tällöin kätkee metodin sen korvaamisen sijaan.

Perusluokan metodin kutsuminen

Jos olet korvannut perusmetodin, on sitä kuitenkin vielä mahdollista kutsua antamalla metodin koko nimi. Se toteutetaan kirjoittamalla perusnimi, kaksi kaksoispistettä ja sitten metodin nimi. Esimerkiksi:

```
Mammal::Move()
```

Listauksen 16.6 rivi 29 voitaisiin nyt kirjoittaa uudelleen muotoon:

```
29:  fido.Mammal::Move(10);
```

Nyt kääntämisen aikaista virhettä ei synny ja Mammal-metodia voidaan kutsua ulkoisesti. Listaus 16.7 havainnollistaa vielä tätä ajattelua.

Listaus 16.7. Perusmetodin kutsuminen korvatust metodista.

```
1:  //Listaus 16.7 Perusmetodin kutsuminen korvatusta metodista
2:
3:  #include <iostream.h>
4:
5:  class Mammal
6:  {
7:  public:
8:    void Move() const { cout << "Mammal move one step\n"; }
9:    void Move(int distance) const
10:     { cout << "Mammal move " << distance << " steps.\n"; }
11: protected:
12:   int itsAge;
13:   int itsWeight;
14: };
15:
16: class Dog : public Mammal
17: {
18: public:
19:   void Move()const;
20: };
21:
22: void Dog::Move() const
23: {
24:   cout << "In dog move...\n";
25:   Mammal::Move(3);
26: }
27:
28: int main()
29: {
30:   Mammal bigAnimal;
31:   Dog fido;
32:   bigAnimal.Move(2);
33:   Fido.Mammal::Move(6);
34:   return 0;
35: }
```

Tulostus

```
Mammal move 2 steps.
Mammal move 6 steps.
```

Analyysi

Rivillä 30 luodaan Mammal-olio, bigAnimal ja rivillä 31 Dog-olio, Fido. Rivillä 32 kutsutaan Mammal-metodia Move(), joka ottaa parametrikseen int-arvon.

Ohjelmoija halusi käyttää Move(int)-metodia Dog-olioon, mutta sai ongelmia. Dog korvaa Move()-metodin, mutta ei ylikuormita sitä eikä siinä ole versiota, joka ottaa int-arvon. Ongelma ratkaistaan kutsumalla ulkoisesti perusluokan Move(int)-metodia (rivi 33).

Tee/Älä tee Laajenna testattujen luokkien toiminnallisuutta johtamalla. Muuta haluttujen funktioiden käyttäytymistä johdetuissa luokissa ylikuormittamalla perusluokkien metodeita.
Älä kätke perusluokan funktiota muuttamalla funktion allekirjoitusta.

Yhteenveto

Tässä luvussa opit, kuinka johdetut luokat perivät ominaisuutensa perusluokilta. Luokat perivät kaikki public- ja protected-tyyppiset tiedot tai funktiot perusluokiltaan.

protected-tyyppinen käyttö on public-tyyppistä johdetuille luokille ja private-tyyppistä kaikille muille olioille. Edes johdetut luokat eivät voi käsitellä perusluokkien private-tyyppistä tietoa tai funktioita.

Muodostimet voidaan alustaa ennen runkoa. Juuri tuossa vaiheessa voidaan käyttää perusmuodostimia hyväksi ja viedä parametreja perusluokille.

Perusluokan funktiot voidaan korvata johdetuissa luokissa. Jos perusluokan funktiot ovat virtuaalisia ja jos olioon viitataan osoittimella tai viittauksella, johdettujen luokkien funktioita voidaan käyttää osoitetun olion ajonaikaiseen tyyppiin perustuen.

Perusluokan metodeja voidaan kutsua nimeämällä funktio siten, että ensin annetaan perusluokan nimi ja sitten kaksi kaksoispistettä ja lopuksi funktion nimi. Esimerkiksi, jos Dog perii metodit Mammal-luokalta, voidaan Mammal-luokan Walk()-metodia kutsua lauseella Mammal::Walk().

Kysymyksiä ja Vastauksia

K
Viedäänkö perityt jäsenet kaikille seuraaville sukupolville? Jos Dog periytyy Mammal-luokasta ja Mammal taas Animal-luokasta, periikö Dog kaikki Animal-luokan jäsenet?
V
Kyllä. Kun johtaminen jatkuu, perivät johdetut luokat kaikki perusluokkiensa jäsenet kumulatiivisesti.

K
Voiko johdettu luokka tehdä public-perusfunktiosta private-tyyppisen?
V
Kyllä ja luokka jää private-tyyppiseksi seuraavissa johtamisissa.