Osa
V

17. oppitunti

Monimuotoisuus ja johdetut luokat

Edellisessä luvussa käsiteltiin periytymistä ja sitä, kuinka johdetut luokat muodostavat periytymishierarkian. Perusluokan metodit voidaan korvata johdetuissa luokissa. Tässä luvussa esitetään, kuinka virtuaalimetodit sallivat perusluokkien käyttämisen monimuotoisesti. Luvussa käsitellään seuraavia aiheita:

- Mitä virtuaalimetodit ovat
- Kuinka virtuaalituhoajaa ja -kopiomuodostinta käytetään
- Virtuaalimetodien käyttämisen vaarat

Virtuaalimetodit

Edellinen luku painotti sitä seikkaa, että Dog-olio on Mammal-olio. Toistaiseksi se on merkinnyt vain sitä, että Dog-olio on perinyt

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

perusluokkansa ominaisuudet ja kyvyt. C++ -kielessä on-suhde menee kuitenkin vielä syvemmälle.

C++ laajentaa monimuotoisuutta sallimalla perusluokkien osoittimien sijoittamisen johdettujen luokkien olioihin. Voimme siis kirjoittaa

```
Mammal * pMammal = new Dog;
```

Lause luo uuden Dog-olion kekoon ja palauttaa osoittimen tuohon olioon, joka sijoitetaan Mammal-osoittimeen. Tämä on hyvä seikka, koska Dog on Mammal.

Huom! Tämä on monimuotoisuuden ydin. Voisit esimerkiksi luoda useita eri tyyppisiä ikkunoita (keskusteluruutuja, vieritettäviä ikkunoita, luetteloruutuja, jne) ja antaa niille kaikille virtuaalisen draw()-metodin. Luomalla osoittimen ikkunaan ja sijoittamalla keskusteluruutuja ja muita johdettuja tyyppejä tuohon osoittimeen voisit kutsua draw()-metodia välittämättä todellisesta ajonaikaisesta oliotyypistä, johon osoitetaan. Oikeaa draw()-metodia kutsuttaisiin aina.

Sen jälkeen voidaan tätä osoitinta käyttää Mammal-metodien kutsumiseen. Nyt Dog-luokassa korvattujen metodien kohdalla voidaan kutsua oikeaa funktiota. Virtuaalifunktiot sallivat sen. Listaus 17.1 havainnollistaa edellä kerrottua ja osoittaa, mitä tapahtuu ei-virtuaalisille metodeille.

Listaus 17.1. Virtuaalimetodien käyttäminen.

```
1:  //Listaus 17.1 Virtuaalimetodit
2:
3:  #include <iostream.h>
4:
5:  class Mammal
6:  {
7:  public:
8:    Mammal():itsAge(1) { cout << "Mammal constructor...\n"; }
9:    ~Mammal() { cout << "Mammal destructor...\n"; }
10:   void Move() const { cout << "Mammal move one step\n"; }
11:   virtual void Speak() const { cout << "Mammal speak!\n"; }
12: protected:
13:   int itsAge;
14:
15: };
16:
17: class Dog : public Mammal
18: {
19: public:
20:   Dog() { cout << "Dog constructor...\n"; }
21:   ~Dog() { cout << "Dog destructor...\n"; }
22:   void WagTail() { cout << "Wagging Tail...\n"; }
23:   void Speak()const { cout << "Woof!\n"; }
24:   void Move()const { cout << "Dog moves 5 steps...\n"; }
25: };
26:
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
27: int main()
28: {
29:
30:   Mammal *pDog = new Dog;
31:   pDog->Move();
32:   pDog->Speak();
33:
34:   return 0;
35: }
```

Tulostus

```
Mammal constructor…
Dog constructor…
Mammal move one step
Woof!
```

Analyysi

Rivillä 11 varustetaan Mammal virtuaalimetodilla Speak(). Luokan suunnittelija viestittää siten, että tämän luokan odotetaan olevan jonkin toisen luokan perusluokkana. Johdettu luokka tulee luultavasti korvaamaan tuon funktion.

Rivillä 30 luodaan Mammal-osoitin, pDog, mutta siihen sijoitetaan uuden Dog-olion osoite. Koska Dog on Mammal, kyseessä on laillinen sijoitus. Osoitinta käytetään sitten Move()-funktion kutsumiseen. Koska kääntäjä tunnistaa pDog-osoittimen Mammal-osoittimeksi, se hakee Move()-metodia Mammal-oliosta.

Rivillä 32 kutsuu osoitin Speak()-metodia. Koska Speak on virtuaalinen, kutsutaan Dog-luokassa korvattua Speak()-metodia.

Tämä on melkein maagista - kutsuvalla funktiolla oli Mammal-osoitin, mutta tässä kutsutaan Dog-metodia. Jos meillä olisi taulukko Mammal-osoittimia, joista jokainen osoittaisi Mammal-aliluokkaan, voitaisiin kutakin osoitinta käyttää vuoron perään ja aina kutsuttaisiin oikeata funktiota. Listaus 17.2 havainnollistaa tätä ajattelua.

Listaus 17.2. Useita virtuaalisia jäsenfunktioita kutsutaan vuorotellen.

```
1:  //Listaus 17.2 Useita virtuaalimetodeja.
2:
3:  #include <iostream.h>
4:
5:  class Mammal
6:  {
7:  public:
8:    Mammal():itsAge(1) {  }
9:    ~Mammal() { }
10:   virtual void Speak() const { cout << "Mammal speak!\n"; }
11: protected:
12:   int itsAge;
13: };
14:
15: class Dog : public Mammal
16: {
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
17: public:
18:   void Speak()const { cout << "Woof!\n"; }
19: };
20:
21:
22: class Cat : public Mammal
23: {
24: public:
25:   void Speak()const { cout << "Meow!\n"; }
26: };
27:
28:
29: class Horse : public Mammal
30: {
31: public:
32:   void Speak()const { cout << "Winnie!\n"; }
33: };
34:
35:
36: class Pig : public Mammal
37: {
38: public:
39:   void Speak()const { cout << "Oink!\n"; }
40: };
41:
42: int main()
43: {
44:   Mammal* theArray[5];
45:   Mammal* ptr;
46:   int choice, i;
47:   for ( i = 0; i<5; i++)
48:   {
49:     cout << "(1)dog (2)cat (3)horse (4)pig: ";
50:     cin >> choice;
51:     switch (choice)
52:     {
53:       case 1: ptr = new Dog;
54:       break;
55:       case 2: ptr = new Cat;
56:       break;
57:       case 3: ptr = new Horse;
58:       break;
59:       case 4: ptr = new Pig;
60:       break;
61:       default: ptr = new Mammal;
62:       break;
63:     }
64:     theArray[i] = ptr;
65:   }
66:   for (i=0;i<5;i++)
67:     theArray[i]->Speak();
68:   return 0;
69: }
```

Tulostus

```
(1)dog (2)cat (3)horse (4)pig:  1
(1)dog (2)cat (3)horse (4)pig:  2
(1)dog (2)cat (3)horse (4)pig:  3
(1)dog (2)cat (3)horse (4)pig:  4
(1)dog (2)cat (3)horse (4)pig:  5
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
Woof!
Meow!
Whinny!
Oink!
Mammal Speak!
```

Analyysi
Tässä karsitussa ohjelmassa on luokilla vain suppea toiminnallisuus ja ohjelma havainnollistaa virtuaalijäsenfunktioiden toimintaa niiden suppeimmassa muodossa. Ohjelmassa esitellään neljä luokkaa: Dog, Cat, Horse ja Pig, jotka on kaikki johdettu Mammal-luokasta.

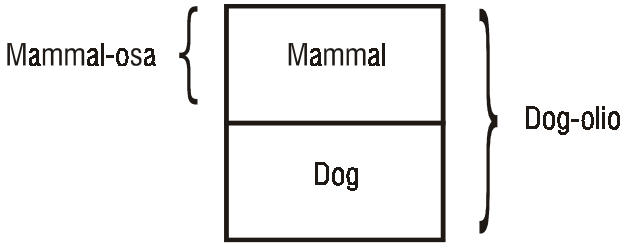
Rivillä 10 esitellään Mammal-luokan Speak()-metodi virtuaalisena. Riveillä 18, 25, 32 ja 38 korvaavat johdetut luokat Speak()-toteutuksen.

Käyttäjää pyydetään kertomaan luotavat oliot ja taulukkoon sijoitetaan osoittimet riveillä 46-63.

Huom! Huomaa, että kääntämisen aikana on mahdotonta tietää, mitä olioita luodaan ja siksi ei tiedetä käytettävää Speak()-metodiakaan. ptr-osoitin sidotaan olioon ajon aikana. Tätä kutsutaan dynaamiseksi sidonnaksi vastakohtana staattiselle sidonnalle tai kääntämisen aikaiselle sidonnalle.

Kuinka virtuaalifunktiot toimivat

Kun johdettu luokka, kuten Dog, luodaan, kutsutaan ensin perusluokan muodostinta ja sitten johdetun luokan muodostinta. Kuva 17.1 esittää, miltä Dog-olio näyttää luomisen jälkeen. Huomaa, että olion Mammal-osa on muistissa heti Dog-osan jälkeen.



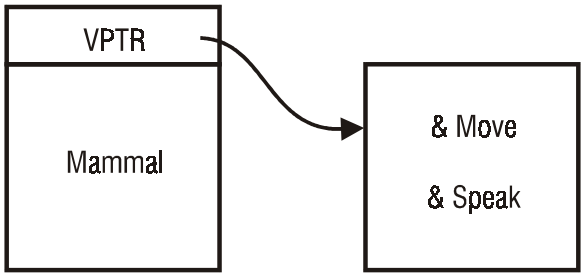
Kuva 17.1. Dog-olio luomisen jälkeen.

Kun virtuaalifunktio luodaan oliossa, täytyy olion valvoa tuota funktiota. Monet kääntäjät muodostavat virtuaalifunktiotaulukon nimeltä v-taulukko. Kullekin tyypille on yksi taulukko ja kukin tuon tyyppinen olio tallentaa virtuaalitaulukko-osoittimen (nimeltä v-osoitin), joka osoittaa tuohon taulukkoon.

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

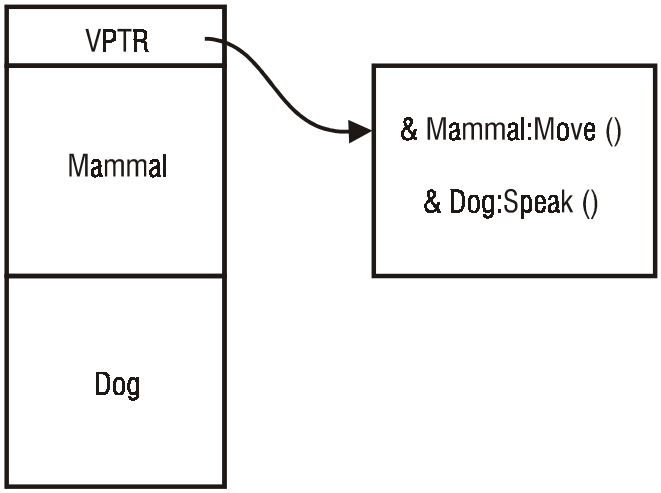
Toteutukset vaihtelevat, mutta kaikki kääntäjät toteuttavat saman asian.

Kunkin olion v-osoitin osoittaa v-taulukkoon, jossa vuorostaan on osoitin kuhunkin virtuaalijäsenfunktioon. Kun Dog-olion Mammal-osa luodaan, alustetaan v-osoitin osoittamaan v-taulukon oikeaan osaan, kuten kuva 17.2 esittää.



Kuva 17.2. Mammal-olion v-taulukko.

Kun Dog-muodostinta kutsutaan ja Dog-osa lisätään, säädetään v-osoitin osoittamaan Dog-oliossa korvattuun virtuaalifunktioon, kuten kuva 17.3 esittää.



Kuva 17.3. Dog-olion v-taulukko.

Kun Mammal-osoitinta käytetään, jatkaa v-osoitin oikeaan funktioon osoittamista riippuen olion todellisesta tyypistä. Kun Speak()-metodia siis kutsutaan, käytetään oikeata funktiota.

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

Et voi siirtyä minne tahansa

Jos Dog-oliolla on metodi WagTail(), joka ei ollut Mammal-luokassa, et voi käyttää Mammal-osoitinta tuon metodin kutsumiseen (ellet muunna sitä Dog-osoittimeksi). Koska WagTail() ei ole virtuaalifunktio ja koska se ei ole Mammal-oliossa, et voi käyttää funktiota ilman Dog-oliota tai Dog-osoitinta.

Vaikka voitkin muuntaa Mammal-osoittimen Dog-osoittimeksi, on yleensä parempiakin ja turvallisempiakin tapoja kutsua WagTail()-metodia. C++ tarjoaa käyttöön muunnokset, koska ne eivät ole virhealttiita. Tätä aihetta käsitellään laajemmin moniperiytyvyyden yhteydessä luvussa 18, "Kehittynyt monimuotoisuus" ja edelleen mallien yhteydessä luvussa 24, "Poikkeukset ja virheiden käsittely".

Pilkkoutuminen

Huomaa, että virtuaalifunktioiden magiikka toimii vain osoittimilla ja viittauksilla. Olion vieminen arvona ei salli virtuaalimetodien käyttöä. Listaus 17.3 havainnollistaa tätä ongelmaa.

Listaus 17.3. Tiedon pilkkoutuminen vietäessä arvona.

```
1:  //Listaus 17.3 Tiedon pirstoutuminen
2:
3:  #include <iostream.h>
4:
5:  enum BOOL { FALSE, TRUE };
6:  class Mammal
7:  {
8:  public:
9:    Mammal():itsAge(1) {  }
10:   ~Mammal() { }
11:   virtual void Speak() const { cout << "Mammal speak!\n"; }
12: protected:
13:   int itsAge;
14: };
15:
16: class Dog : public Mammal
17: {
18: public:
19:   void Speak()const { cout << "Woof!\n"; }
20: };
21:
22: class Cat : public Mammal
23: {
24: public:
25:   void Speak()const { cout << "Meow!\n"; }
26: };
27:
28: void ValueFunction (Mammal);
29: void PtrFunction (Mammal*);
30: void RefFunction (Mammal&);
31: int main()
32: {
33:   Mammal* ptr=0;
34:   int choice;
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
35:   while (1)
36:   {
37:     BOOL fQuit = FALSE;
38:     cout << "(1)dog (2)cat (0)Quit: ";
39:     cin >> choice;
40:     switch (choice)
41:     {
42:       case 0: fQuit = TRUE;
43:       break;
44:       case 1: ptr = new Dog;
45:       break;
46:       case 2: ptr = new Cat;
47:       break;
48:       default: ptr = new Mammal;
49:       break;
50:     }
51:     if (fQuit)
52:       break;
53:     PtrFunction(ptr);
54:     RefFunction(*ptr);
55:     ValueFunction(*ptr);
56:   }
57:   return 0;
58: }
59:
60: void ValueFunction (Mammal MammalValue)
61: {
62: MammalValue.Speak();
63: }
64:
65: void PtrFunction (Mammal * pMammal)
66: {
67:   pMammal->Speak();
68: }
69:
70: void RefFunction (Mammal & rMammal)
71: {
72:   rMammal.Speak();
73: }
```

Tulostus

```
(1)dog (2)cat (0)Quit: 1
woof!
Woof!
Mammal Speak!
(1)dog (2)cat (0)Quit: 2
Meow!
Meow!
Mammal Speak!
(1)dog (2)cat (0)Quit: 0
```

Analyysi

Rivillä 6-27 esitellään karsitut versiot Mammal-, Dog- ja Cat-luokista. Listauksessa esitellään kolme funktiota, PtrFunction(), RefFunction() ja ValueFunction(). Ne ottavat parametreikseen osoittimen Mammal-olioon, Mammal-viittauksen ja Mammal-olion. Kaikki kolme funktiota tekevät saman asia, ne kutsuvat Speak()-metodia.

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

Käyttäjää pyydetään valitsemaan Dog tai Cat; valinnan mukaan luodaan osoitin vastaavaan kohteeseen riveillä 44-49.

Tulostuksen alussa käyttäjä valitsee Dogin. Dog-olio luodaan vapaaseen tilaan rivillä 44. Sitten Dog viedään osoittimena, viittauksena ja arvona kolmelle funktiolle.

Osoitin ja viittaukset liittyvät virtuaalisiin jäsenfunktioihin ja Dog->Speak()-metodia kutsutaan. Tulostus osoittaa tämän.

Uudelleen viitattu osoitin viedään kuitenkin arvona. Funktio odottaa Mammal-oliota, joten kääntäjä pilkkoo Dog-oliosta pelkän Mammal-osan. Nyt kutsutaan Mammalin Speak()-metodia, kuten tulostuksen kolmas rivi kertoo.

Tämä kokeilu toistetaan sitten Cat-oliolle samanlaisin tuloksin.

Virtuaaliset tuhoajafunktiot

On laillista ja yleistä viedä osoitin johdettuun olioon, kun odotetaan osoitinta perusolioon. Mitä tapahtuu, kun osoitin johdettuun kohteeseen tuhotaan? Jos tuhoajafunktio on virtuaali, kuten sen tulisi olla, tapahtuu oikea asia - kutsutaan johdetun luokan tuhoajafunktiota. Koska johdetun luokan tuhoajafunktio käyttää automaattisesti perusluokan tuhoajafunktiota, tuhotaan koko kohde oikealla tavalla.

Peukalosääntö on seuraava: Jos jokin luokkasi funktio on virtuaalinen, tulisi tuhoajafunktionkin olla sellainen.

Virtuaaliset kopiomuodostimet

Kuten aiemmin esitettiin, ei mikään muodostin voi olla virtuaali. Kuitenkin on tilanteita, jolloin ohjelmasi on kyettävä viemään osoitin peruskohteeseen ja tehdä kopio oikeasta johdetusta kohteesta, joka luodaan. Yleinen ratkaisu ongelmaan on luoda kloonimetodi perusluokkaan ja tehdä siitä virtuaali. Kloonimetodi luo uuden kopion nykyisestä oliosta ja palauttaa tuon olion.

Koska jokainen johdettu luokka korvaa kloonimetodin, luodaan kopio johdetusta luokasta. Listaus 14.7 havainnollistaa edellä kerrottua.

Listaus 17.4. Virtuaali kopiomuodostin.

```
1:  //Listaus 17.4 Virtuaaliset kopiomuodostimet
2:
3:  #include <iostream.h>
4:
5:  class Mammal
6:  {
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
7:  public:
8:    Mammal():itsAge(1) { cout << "Mammal constructor...\n"; }
9:    ~Mammal() { cout << "Mammal destructor...\n"; }
10:   Mammal (const Mammal & rhs);
11:   virtual void Speak() const { cout << "Mammal speak!\n"; }
12:   virtual Mammal* Clone() { return new Mammal(*this); }
13:   int GetAge()const { return itsAge; }
14: protected:
15:   int itsAge;
16: };
17:
18: Mammal::Mammal (const Mammal & rhs):itsAge(rhs.GetAge())
19: {
20:   cout << "Mammal Copy Constructor...\n";
21: }
22:
23: class Dog : public Mammal
24: {
25: public:
26:   Dog() { cout << "Dog constructor...\n"; }
27:   ~Dog() { cout << "Dog destructor...\n"; }
28:   Dog (const Dog & rhs);
29:   void Speak()const { cout << "Woof!\n"; }
30:   virtual Mammal* Clone() { return new Dog(*this); }
31: };
32:
33: Dog::Dog(const Dog & rhs):
34: Mammal(rhs)
35: {
36:   cout << "Dog copy constructor...\n";
37: }
38:
39: class Cat : public Mammal
40: {
41: public:
42:   Cat() { cout << "Cat constructor...\n"; }
43:   ~Cat() { cout << "Cat destructor...\n"; }
44:   Cat (const Cat &);
45:   void Speak()const { cout << "Meow!\n"; }
46:   virtual Mammal* Clone() { return new Cat(*this); }
47: };
48:
49: Cat::Cat(const Cat & rhs):
50: Mammal(rhs)
51: {
52:   cout << "Cat copy constructor...\n";
53: }
54:
55: enum ANIMALS { MAMMAL, DOG, CAT};
56: const int NumAnimalTypes = 3;
57: int main()
58: {
59:   Mammal *theArray[NumAnimalTypes];
60:   Mammal* ptr;
61:   int choice,i;
62:   for (i = 0; i<NumAnimalTypes; i++)
63:   {
64:     cout << "(1)dog (2)cat (3)Mammal: ";
65:     cin >> choice;
66:     switch (choice)
67:     {
```

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

```
68:       case DOG: ptr = new Dog;
69:       break;
70:       case CAT: ptr = new Cat;
71:       break;
72:       default: ptr = new Mammal;
73:       break;
74:     }
75:     theArray[i] = ptr;
76:   }
77:   Mammal *OtherArray[NumAnimalTypes];
78:   for (i=0;i<NumAnimalTypes;i++)
79:   {
80:     theArray[i]->Speak();
81:     OtherArray[i] = theArray[i]->Clone();
82:   }
83:   for (i=0;i<NumAnimalTypes;i++)
84:     OtherArray[i]->Speak();
85:   return 0;
86: }
```

Tulostus

```
1:  (1)dog (2)cat (3)Mammal:1
2:  Mammal constructor…
3:  Dog constructor…
4: (1)dog (2)cat (3)Mammal:2
5: Mammal constructor…
6:  Cat constructor…
7: (1)dog (2)cat (3)Mammal:3
8: Mammal constructor…
9:  Woof!
10: Mammal copy constructor…
11: Dog copy constructor…
12: Meow!
13: Mammal copy constructor…
14: Cat copy constructor…
15: Mammal speak!
16: Mammal copy constructor…
17: Woof!
18: Meow!
19: Mammal speak!
```

Analyysi

Listaus 17.4 on hyvin samanlainen kuin aiemmat kaksi listausta paitsi, että Mammal-luokkaan on lisätty uusi virtuaalimetodi: clone(). Tämä metodi palauttaa osoittimen uuteen Mammal-olioon kutsumalla kopiomuodostinta, vieden parametrina itsensä (*this) kopioviittauksena.

Dog ja Cat korvaavat clone()-metodin alustaen tietonsa ja vieden kopion itsestään omille kopiomuodostimilleen. Koska clone() on virtuaali, luodaan virtuaali kopiomuodostin, kuten rivi 81 osoittaa.

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

Käyttäjää kehotetaan valitsemaan koiria, kissoja tai elukoita ja ne luodaan riveillä 62-74. Osoitin kuhunkin valintaan tallennetaan taulukkoon rivillä 75.

Lopuksi on kunkin olion Speak()- ja Clone()-metodeita kutsuttu riveillä 80-81. Clone()-kutsun tuloksena on osoitin olion kopioon, joka tallennetaan sitten toiseen taulukkoon rivillä 81.

Tulostusrivillä 1 on käyttäjä valinnut vaihtoehdon 1 eli koiran. Nyt käytetään Mammal- ja Dog-muodostimia. Sama toistuu Cat- ja Mammal-olioille muodostimen riveillä 4-8.

Tulostusrivi 9 esittää ensimmäisen olion, Dog-olion, Speak()-kutsua. Nyt kutsutaan virtuaalia Speak()-metodia ja käytetään oikeata Speak()-versiota. Seuraavaksi kutsutaan Clone()-funktiota ja koska sekin on virtuaali, käytetään Dogin Clone-metodia, jolloin kutsutaan Mammal-muodostinta ja Dogin kopiomuodostinta.

Sama toistuu Cat-oliolle riveillä 12-14 ja sitten Mammal-oliolle riveillä 15-16. Lopuksi käytetään uutta taulukkoa ja kutsutaan jokaisen uuden olion Speak()-metodia.

Virtuaalimetodien huonot puolet

Koska virtuaalimetodeja sisältävien olioiden tulee ylläpitää v-taulukkoa, aiheuttavat virtuaalimetodit siis hieman ylimääräistä työtä. Jos sinulla on hyvin pieni luokka, josta ei johdeta uusia luokkia, on monia syitä jättää virtuaalimetodit pois.

Kun esittelet metodin virtuaalina, on jo maksettu suurin osa v-taulukon hinnasta (vaikkakin kukin tieto aiheuttaa hieman lisää muistin käyttöä). Yleensä tuhoajafunktiosta halutaan ensin tehdä virtuaali ja muiden metodien oletetaan olevan myöskin virtuaaleja. Tutki kuitenkin mitä tahansa ei-virtuaalia metodia ja varmista, että ymmärrät, miksi ne eivät ole virtuaaleja.

Tee/Älä tee

Käytä virtuaalimetodeja, kun odotat luokkaa käytettävän johtamiseen.
Käytä virtuaalituhoajaa, jos jokin metodi on virtuaali.
Älä tee muodostimesta virtuaalia.

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

Yhteenveto

Tässä luvussa opit käyttämään virtuaaleja jäsenfunktioita, joilla sallitaan johdettujen luokkien käyttäytyvän monimuotoisesti perusluokkiensa suhteen.

Jos luokassa on virtuaalimetodeja, tuhoajafunktion tulisi myös olla virtuaalin. Virtuaali tuhoajafunktio takaa sen, että olion johdettu osa vapautetaan käytettäessä delete-operaattoria osoittimelle. Muodostimet eivät voi olla virtuaaleja. Virtuaaleja kopiomuodostimia voidaan luoda tehokkaasti luomalla virtuaali jäsenfunktio, joka kutsuu kopiomuodostinta.

Kysymyksiä ja Vastauksia

K
Miksi kaikista luokan funktiosta ei tehdä virtuaaleja?
V
Virtuaalifunktio vie tehoa v-taulukon muodossa. Jos yksi funktio on virtuaali, tulisi niitä olla useampikin juuri tuon v-taulukon luomisen takia.

K
Jos funktio, SomeFunc(), on virtuaali perusluokassa ja on ylikuormitettu ottamaan parametreikseen joko yhden tai kaksi kokonaislukua ja johdettu luokka korvaa funktion ottamaan yhden kokonaisluvun, mitä kutsutaan, kun osoitin johdettuun olioon kutsuu kaksi kokonaislukua ottavaa funktiotyyppiä?
V
Korvaava yhden int-arvon ottava funktio kätkee perusluokan funktion: siten nyt saadaan kääntäjän virhe, joka valittaa, että funktio vaatii vain yhden int-arvon.

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.

Copyright © IT Press – Tämän e-kirjan kopiointi, tulostaminen ja jakeleminen eteenpäin luvatta on kielletty.