Pewarisan (Inheritance)

Pertemuan 10

Outline

- Inheritance
  - The Syntax of Derivation
  - Private Versus Protected
  - Constructors and Destructors
  - Passing Arguments to Base Constructors
  - Overriding Functions

Dasar Teori

- Inheritance atau pewarisan merupakan salah satu keistimewaan dalam pemrograman berorientasi objek. Inheritance mengijinkan kita untuk membuat objek yang diturunkan dari objek lain, sehingga dimungkinkan didalamnya terdapat member lain selain membernya sendiri.
- Contoh : Jika kita ingin mendeklarasikan 2 buah class yang mendeskripsikan polygon yaitu CRectangle atau CTriangle. Keduanya dapat dideskripsikan dengan dua atribut yakni : alas dan tinggi. Hal ini dapat direpresentasikan dengan class CPolygon dan dari class tersebut diturunkan dua class yakni CRectangle dan CTriangle.
- Class CPolygon berisi member yang umum pada semua polygon, dalam contoh ini adalah panjang dan lebar (width dan height).

Dasar Teori

- CRectangle dan CTriangle diturunkan dari class tersebut. Class yang diturunkan dari class lain mewarisi semua member yang ada dalam class dasarnya.
- Hal ini berarti bahwa jika class dasarnya memiliki member A dan kita menurunkannya ke class lain yang memiliki member B, maka class turunan akan terdiri dari A dan B.
- Untuk menurunkan class dari yang lain, kita menggunakan operator : (colon) dalam deklarasi class turunan dengan cara sebagai berikut:

class derived_class_name: public base_class_name;

- Dimana derived_class_name adalah nama class turunan dan base_class_name adalah nama class yang menjadi dasar. Public dapat diganti dengan akses lain misalnya protected atau private, dan menjelaskan akses untuk member yang diturunkan

Dasar Teori

- Karakterisitik akses member sebuah objek.

Access	public	protected	private
Anggota dari kelas yang sama	yes	yes	yes
Anggota dari turunan kelas	yes	yes	no
Bukan anggota	yes	no	no

1. The Syntax of Derivation

Mammal1.cpp

```
1: #include <iostream>
2:
3: enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
4:
5: class Mammal
6: {
7:        public:
8:                  // constructors
9:                  Mammal();
10:                 ~Mammal();
11:
12:                 // accessors
13:                 int getAge() const;
14:                 void setAge(int);
15:                 int getWeight() const;
16:                 void setWeight();
17:
18:                 // other methods
19:                 void speak();
20:                 void sleep();
21:
22:       protected:
23:                 int age;
24:                 int weight;
25: };
26:
```

...

```
27: class Dog : public Mammal
28: {
29:       public:
30:                 // constructors
31:                 Dog();
32:                 ~Dog();
33:
34:                 // accessors
35:                 BREED getBreed() const;
36:                 void setBreed(BREED);
37:
38:                 // other methods
39:                 // wagTail();
40:                 // begForFood();
41:
42:       protected:
43:                 BREED breed;
44: };
45:
46: int main()
47: {
48:       return 0;
49: }
```

2. Private Versus Protected

Mammal2.cpp

```
1: #include <iostream>
2:
3: enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
4:
5: class Mammal
6: {
7:        public:
8:                  // constructors
9:                  Mammal(): age(2), weight(5) {}
10:                 ~Mammal(){}
11:
12:                 // accessors
13:                 int getAge() const { return age; }
14:                 void setAge(int newAge) { age = newAge; }
15:                 int getWeight() const { return weight; }
16:                 void setWeight(int newWeight) { weight = newWeight; }
17:
18:                 // other methods
19:                 void speak() const { std::cout << "Mammal sound!\n"; }
20:                 void sleep() const { std::cout << "Shhh. I'm sleeping.\n"; }
21:
22:       protected:
23:                 int age;
24:                 int weight;
25: };
26:
```

…

```
27: class Dog : public Mammal
28: {
29:       public:
30:                 // constructors
31:                 Dog(): breed(YORKIE) {}
32:                 ~Dog() {}
33:
34:                 // accessors
35:                 BREED getBreed() const { return breed; }
36:                 void setBreed(BREED newBreed) { breed = newBreed; }
37:
38:                 // other methods
39:                 void wagTail() { std::cout << "Tail wagging ...\n"; }
40:                 void begForFood() { std::cout << "Begging for food ...\n"; }
41:
42:       private:
43:                 BREED breed;
44: };
45:
46: int main()
47: {
48:       Dog fido;
49:       fido.speak();
50:       fido.wagTail();
51:       std::cout << "Fido is " << fido.getAge() << " years old\n";
52:       return 0;
53: }
```

3. Constructors and Destructors

Mammal3.cpp

```
1: #include <iostream>
2:
3: enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
4:
5: class Mammal
6: {
7:         public:
8:                   // constructors
9:                   Mammal();
10:                  ~Mammal();
11:
12:                  // accessors
13:                  int getAge() const { return age; }
14:                  void setAge(int newAge) { age = newAge; }
15:                  int getWeight() const { return weight; }
16:                  void setWeight(int newWeight) { weight = newWeight; }
17:
18:                  // other methods
19:                  void speak() const { std::cout << "Mammal sound!\n"; }
20:                  void sleep() const { std::cout << "shhh. I'm sleeping.\n"; }
21:
22:        protected:
23:                  int age;
24:                  int weight;
25: };
26:
```

...

```
27: class Dog : public Mammal
28: {
29:       public:
30:                 // constructors
31:                 Dog();
32:                 ~Dog();
33:
34:                 // accessors
35:                 BREED getBreed() const { return breed; }
36:                 void setBreed(BREED newBreed) { breed = newBreed; }
37:
38:                 // other methods
39:                 void wagTail() { std::cout << "Tail wagging ...\n"; }
40:                 void begForFood() { std::cout << "Begging for food ...\n"; }
41:
42:       private:
43:                 BREED breed;
44: };
45:
46: Mammal::Mammal():
47: age(1),
48: weight(5)
49: {
50:       std::cout << "Mammal constructor ...\n";
51: }
52:
```

...

```
53: Mammal::~Mammal()
54: {
55:       std::cout << "Mammal destructor ...\n";
56: }
57:
58: Dog::Dog():
59: breed(YORKIE)
60: {
61:       std::cout << "Dog constructor ...\n";
62: }
63:
64: Dog::~Dog()
65: {
66:       std::cout << "Dog destructor ...\n";
67: }
68:
69: int main()
70: {
71:       Dog fido; // create a dog
72:       fido.speak();
73:       fido.wagTail();
74:       std::cout << "Fido is " << fido.getAge() << " years old\n";
75:       return 0;
76: }
```

4. Passing Arguments to Base Constructors

Mammal4.cpp

```
1: #include <iostream>
2:
3: enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
4:
5: class Mammal
6: {
7:        public:
8:                  // constructors
9:                  Mammal();
10:                 Mammal(int age);
11:                 ~Mammal();
12:
13:                 // accessors
14:                 int getAge() const { return age; }
15:                 void setAge(int newAge) { age = newAge; }
16:                 int getWeight() const { return weight; }
17:                 void setWeight(int newWeight) { weight = newWeight; }
18:
19:                 // other methods
20:                 void speak() const { std::cout << "Mammal sound!\n"; }
21:                 void sleep() const { std::cout << "Shhh. I'm sleeping.\n"; }
22:
23:       protected:
24:                 int age;
25:                 int weight;
26: };
27:
```

...

```
28: class Dog : public Mammal
29: {
30:         public:
31:                     // constructors
32:                     Dog();
33:                     Dog(int age);
34:                     Dog(int age, int weight);
35:                     Dog(int age, BREED breed);
36:                     Dog(int age, int weight, BREED breed);
37:                     ~Dog();
38:
39:                     // accessors
40:                     BREED getBreed() const { return breed; }
41:                     void setBreed(BREED newBreed) { breed = newBreed; }
42:
43:                     // other methods
44:                     void wagTail() { std::cout << "Tail wagging ...\n"; }
45:                     void begForFood() { std::cout << "Begging for food ...\n"; }
46:
47:         private:
48:                     BREED breed;
49: };
50:
51: Mammal::Mammal():
52: age(1),
53: weight(5)
54: {
55:         std::cout << "Mammal constructor ...\n";
56: }
57:
```

...

```
58: Mammal::Mammal(int age):
59: age(age),
60: weight(5)
61: {
62:         std::cout << “Mammal(int) constructor ...\n”;
63: }
64:
65: Mammal::~Mammal()
66: {
67:         std::cout << “Mammal destructor ...\n”;
68: }
69:
70: Dog::Dog():
71: Mammal(),
72: breed(YORKIE)
73: {
74:         std::cout << “Dog constructor ...\n”;
75: }
76:
77: Dog::Dog(int age):
78: Mammal(age),
79: breed(YORKIE)
80: {
81:         std::cout << “Dog(int) constructor ...\n”;
82: }
83:
84: Dog::Dog(int age, int newWeight):
85: Mammal(age),
86: breed(YORKIE)
87: {
88:         weight = newWeight;
89:         std::cout << “Dog(int, int) constructor ...\n”;
90: }
91:
```

...

```
92: Dog::Dog(int age, int newWeight, BREED breed):
93: Mammal(age),
94: breed(breed)
95: {
96:         weight = newWeight;
97:         std::cout << "Dog(int, int, BREED) constructor ...\n";
98: }
99:
100: Dog::Dog(int age, BREED newBreed):
101: Mammal(age),
102: breed(newBreed)
103: {
104:        std::cout << "Dog(int, BREED) constructor ...\n";
105: }
106:
107: Dog::~Dog()
108: {
109:        std::cout << "Dog destructor ...\n";
110: }
111:
112: int main()
113: {
114:        Dog fido;
115:        Dog rover(5);
116:        Dog buster(6, 8);
117:        Dog yorkie (3, YORKIE);
118:        Dog dobbie (4, 20, DOBERMAN);
119:        fido.speak();
120:        rover.wagTail();
121:        std::cout << "Yorkie is "
122:        << yorkie.getAge() << " years old\n";
123:        std::cout << "Dobbie weighs "
124:        << dobbie.getWeight() << " pounds\n";
125:        return 0;
126: }
```

5. Overriding Functions

Mammal5.cpp

```
1: #include <iostream>
2:
3: enum BREED { YORKIE, CAIRN, DANDIE, SHETLAND, DOBERMAN, LAB };
4:
5: class Mammal
6: {
7:        public:
8:                  // constructors
9:                  Mammal() { std::cout << "Mammal constructor ...\n"; }
10:                 ~Mammal() { std::cout << "Mammal destructor ...\n"; }
11:
12:                 // other methods
13:                 void speak() const { std::cout << "Mammal sound!\n"; }
14:                 void sleep() const { std::cout << "Shhh. I'm sleeping.\n"; }
15:
16:       protected:
17:                 int age;
18:                 int weight;
19: };
20:
```

...

```
21: class Dog : public Mammal
22: {
23:       public:
24:                 // constructors
25:                 Dog() { std::cout << "Dog constructor ...\n"; }
26:                 ~Dog() { std::cout << "Dog destructor ...\n"; }
27:
28:                 // other methods
29:                 void wagTail() { std::cout << "Tail wagging ...\n"; }
30:                 void begForFood() { std::cout << "Begging for food ...\n"; }
31:                 void speak() const { std::cout << "Woof!\n"; }
32:
33:       private:
34:                 BREED breed;
35: };
36:
37: int main()
38: {
39:                 Mammal bigAnimal;
40:                 Dog fido;
41:                 bigAnimal.speak();
42:                 fido.speak();
43:                 return 0;
44: }
```

6. Hiding the Base Class Method

Mammal6.cpp

```
1: #include <iostream>
2:
3: class Mammal
4: {
5:         public:
6:                   void move() const { std::cout << "Mammal moves one step\n"; }
7:                   void move(int distance) const
8:                   { std::cout << "Mammal moves " << distance <<" steps\n"; }
9:         protected:
10:                  int age;
11:                  int weight;
12: };
13:
14: class Dog : public Mammal
15: {
16:        public:
17:        void move() const { std::cout << "Dog moves 5 steps\n"; }
18: }; // you may receive a warning that you are hiding a function!
19:
20: int main()
21: {
22:        Mammal bigAnimal;
23:        Dog fido;
24:        bigAnimal.move();
25:        bigAnimal.move(2);
26:        fido.move();
27:        // fido.move(10);
28:        return 0;
29: }
```

7. Calling the Base Method

Mammal7.cpp

```
1: #include <iostream>
2:
3: class Mammal
4: {
5:          public:
6:                     void move() const { std::cout << "Mammal moves one step\n"; }
7:                     void move(int distance) const
8:                     { std::cout << "Mammal moves " << distance << " steps\n"; }
9:          protected:
10:                    int age;
11:                    int weight;
12: };
13:
14: class Dog : public Mammal
15: {
16:         public:
17:         void move() const;
18: };
19:
20: void Dog::move() const
21: {
22:         std::cout << "Dog moves ...\n";
23:         Mammal::move(3);
24: }
25:
26: int main()
27: {
28:         Mammal bigAnimal;
29:         Dog fido;
30:         bigAnimal.move(2);
31:         fido.Mammal::move(6);
32:         return 0;
33: }
```

Tugas

- Pada program Mammal6.cpp, lakukan uncomment pada baris ke 28. Apa yang terjadi? Apa yang harus Anda lakukan agar membuat program ini bekerja?
- Modifikasi program Mammal2.cpp untuk menggunakan sebuah string text dibandingkan sebelumnya menggunakan tipe data enumerated sebagai breed.