آموزش C++ - درس ۹: مقدمه ای بر کلاسها

تاکنون، آموزشهای ما بر ارائه اصول نحوی C++ تاکید داشت. شما چگونگی تعریف و استفاده از متغیرها، کنترل جریان اجرای برنامه با استفاده از حلقه ها و پردازش شرطی ، برخی راههای اساسی برای مدیریت ورودی و خروجی، اشاره گرها و آرایه ها را یاد گرفتید. این درس کلاسها و اشیا را به شما معرفی می کند. استفاده از اشیا در C++ مسیری برای طراحی و نوشتن برنامه ها تعیین کرد. کلاسها یک ایجاد کننده نرم افزار هستند که می توانند برای مدلسازی اشیای دنیای واقعی استفاده شوند. کلاسها داده ها و توانایی های آنها را کپسوله می کنند. برای مثال، یک مدل نرم افزار از یک ماشین، یک کلاس ماشین، ممکن است حاوی داده هایی درباره نوع ماشین و توانایی هایی مانند شتاب و سرعت باشد. یک کلاس یک نوع داده تعریف شده توسط برنامه نویس است که دارای داده ، اعضای آن و توانایی ها و متدهایش می باشد. یک شی یک نمونه خاص از یک کلاس است. این بهترین مثال بدیهی است بوسیله یک مقایسه با یکی از انواع داده داخلی مانند int.

```
int x;
```

متغیر x را از نوع int تعریف کرده است.

```
Car impala;
```

Impala را به عنوان یک شی از کلاس Car تعریف کرده است.

توجه: مفاهیم مطرح شده در این درس در درسهای آینده با جزییات بیشتر دوباره مطرح خواهند شد.

تعریف کلاسها

یک کلاس با استفاده از لغت کلیدی class بوسیله نام تعیین شده توسط برنامه نویس و با استفاده از آکولادها تعریف می شود. تعریف کلاس حاوی اعضای کلاس به عنوان داده های آن و متدهای کلاس به عنوان توابع آن است. به عنوان مثال، اجازه دهید یک کلاس Dog تعریف کنیم که یک مدل از حیوانات خانگی است که بسیاری از ما داریم.

```
class Dog {
public:
    void setAge(int age);
    int getAge();
    void setWeight(int weight);
    int getWeight();
    void speak();
private:
    int age;
    int weight;
};
```

این مثال ساده چندین مفهوم مهم را نشان می دهد. ابتدا، لغت کلیدی private به این نکته اشاره دارد که دو عضو age و weight نمی توانند بطور مستقیم از بیرون کلاس مورد دسترسی قرار گیرند. لغت کلیدی public دلالت بر این دارد که متدهای setAge و getAge و setWeight و getWeight و speak را می توان از بیرون از کلاس فراخوانی کرد. آنها ممکن است با استفاده از اشیایی از این کلاس از قسمتهای دیگر یک برنامه فراخوانی شوند. این تکنیک که دسترسی و تغییرات اعضای داده را فقط از طریق متدها اجازه می دهد مربوط به اصل پنهان سازی داده هاست. رابط کلاس public و داده private است. رابط public و داده private یک مفهوم کلیدی در زمان طراحی کردن کلاسها است. پنهان سازی داده در قسمت دیگری در همین درس مورد بررسی قرار خواهد گرفت. همچنین توجه کنید که ۴ متد setAge و getAge و setWeight و getWeight با خواندن و یا بروز در آوردن مقدار اعضای داده کلاس سر و کار دارند. متدهای استفاده شده برای مقداردهی و یا بدست آوردن مقدار اعضای داده، متدهای دسترسی داده یا متدهای دسترسی نامیده می شوند.

در تعریف کلاس فوق، متدها اعلان شدند ولی تعریف نشدند. آن یک پیاده سازی برای هر متد است که باید نوشته شود.

```
class Dog {
public:
    void setAge(int age);
    int getAge();
    void setWeight(int weight);
```

```
    int getWeight();
    void speak();
private:
    int age;
    int weight;
};

void Dog::setAge(int age)
{
    this->age = age;
}

int Dog::getAge()
{
    return age;
}

void Dog::setWeight(int weight)
{
    this->weight = weight;
}

int Dog::getWeight()
{
    return weight;
}

void Dog::speak()
{
    cout << "BARK!!" << endl;
}
```

در اینجا چند چیز مهم است که باید به آنها توجه کنیم. ابتدا، از آنجایی که تعریف متدها در خارج از کلاس قرار دارد آنها باید بوسیله آن کلاس تشخیص داده شوند. اینکار با عملگر حوزه تفکیک "::" انجام می شود. آن هر متد را شناسایی می کند برای مثال، getAge متعلق به کلاس Dog است. دوم، هر شی یک اشاره گر مخصوص دارد که this نامیده می شود، که مربوط به خود شی است (به خود شی ارجاع می کند). بنابراین اعضای کلاس Dog می توانند به this->age یا this->weight به همان خوبی age یا weight مراجعه کنند. اگر هیچ ابهامی وجود ندارد، نیازی به قید آن نیست. پس در متد setWeight به جای استفاده از this->weight می توانید از weight استفاده کنید. در متد setWeight یک ابهام وجود دارد. از آنجایی که پارامتر ارسال شده weight نام دارد و در اینجا یک عضو کلاس weight نام دارد، اشاره گر this باید استفاده شود. و در آخر نکته ای در مورد نحو زبان. اگر this اشاره گری به یک کلاس است، سپس عملگر انتخاب عضو "<-" برای دسترسی به اعضای آن می تواند مورد استفاده قرار گیرد.

سازنده ها و مخربها

هر کلاس اغلب یک متد خاص دارد به نام سازنده که وقتی یک شی از آن کلاس ساخته می شود فراخوانی می گردد. متد سازنده می تواند برای مقداردهی اولیه متغیرها، تخصیص حافظه پویا یا آماده سازی هر نوع منبع استفاده شود. متد خاص دیگر، متد مخرب، است که وقتی یک شی در حال از بین رفتن است فراخوانی می شود. یک شی وقتی از بین می رود که از میدان دید خارج شود. اگر یک شی درون یک تابع ساخته شده باشد، آن شی وقتی که تابع به پایان رسید از حوزه دید خارج می شود. از آنجایی که برنامه شما تابع main است، وقتی که برنامه به پایان می رسد همه اشیا از حوزه دید خارج می شوند. حوزه دید بطور کامل در درس بعدی توضیح داده می شود. متد مخرب برای آزاد کردن هر حافظه ای که تخصیص داده شده و آزاد کردن منابع دیگر بکار می رود. در اینجا به کلاس Dog یک متد سازنده و یک متد مخرب اضافه شده است. در اینجا اعضای تابعی (توابع عضو کلاس) دوباره نوشته نشده اند.

```
class Dog {
public:
    Dog();      //Constructor
    ~Dog();     //Destructor
    void setAge(int age);
    int getAge();
    void setWeight(int weight);
    int getWeight();
    void speak();
private:
    int age;
    int weight;
};
```

```
Dog::Dog()
{
   age = 0;
   weight = 0;
   cout << "Dog Constructor Called" << endl;
}

Dog::~Dog()
{
   cout << "Dog Destructor Called" << endl;
}
```

توجه کنید که نام متد سازنده همان نام کلاس است. متد مخرب همان نام کلاس را به همراه پیشوند ~ دارد. در مثال قبل متد سازنده برای مقداردهی اولیه متغیرهای عضو بکار رفته است. در کلاسهای دیگر متد سازنده ممکن است تخصیص حافظه، بدست آوردن کنترل منابع مانند ابزارهای سیستم یا اجرا کردن کدهای مقداردهی اولیه پیچیده را انجام دهد. متد مخرب در مثال بالا یک عملیات واقعی را انجام نداده است، فقط پیغام اجرا شدن آن را نمایش می دهد. در کلاسهای دیگر متد مخرب ممکن است آزاد کردن حافظه ای که تخصیص یافته است، آزاد کردن منابع و یا نمایش بعضی از عملیاتهای پاک کردن. همانطور که در درسهای بعدی خواهید دید، داشتن چندین متد سازنده که در تعداد و (یا) نوع پارامترها متفاوت هستند نیز امکان پذیر است. متد سازنده ای که به عنوان مبنا استفاده می شود بر پایه آرگومانهای بکار رفته انتخاب می شود. این مبحث را در قسمت method overloading بررسی خواهیم کرد.

بکار بردن اشیا

برای کمک به درک نحوه استفاده از اشیا، برنامه زیر اشیایی را از کلاس Dog تعریف کرده است. برای سادگی، همه کد در یک فایل منبع تکی قرار گرفته است، البته در پروژه های بزرگتر کلاسها معمولا در فایلهای جداگانه ای از برنامه اصلی نگهداری می شوند.

```
#include <iostream>
using namespace std;

class Dog {
private:
   int age;
   int weight;
public:
   Dog();      //Constructor
   ~Dog();    //Destructor
   void setAge(int age);
   int getAge();
   void setWeight(int weight);
   int getWeight();
   void speak();
};

Dog::Dog()
{
   age = 0;
   weight = 0;
   cout << "Dog Constructor Called" << endl;
}

Dog::~Dog()
{
   cout << "Dog Destructor Called" << endl;
}

void Dog::setAge(int age)
{
   this->age = age;
}

int Dog::getAge()
{
   return age;
}

void Dog::setWeight(int weight)
{
```

```
    this->weight = weight;
}

int Dog::getWeight()
{
    return weight;
}

void Dog::speak()
{
    cout << "BARK!!" << endl;
}

int main()
{
    Dog fido;
    Dog rover;

    cout << "Rover is " << rover.getAge() << " years old." << endl;
    cout << "He weighs " << rover.getWeight() << " lbs." << endl;
    cout << endl;

    cout << "Updating Rover's Age and Weight" << endl;
    rover.setAge(1);
    rover.setWeight(10);

    cout << "Rover is " << rover.getAge() << " years old." << endl;
    cout << "He weighs " << rover.getWeight() << " lbs." << endl;
    cout << endl;

    cout << "Fido is " << fido.getAge() << " years old." << endl;
    cout << "He weighs " << fido.getWeight() << " lbs." << endl;

    cout << "Setting Fido to be the same as Rover" << endl;
    fido = rover;

    cout << "Fido is " << fido.getAge() << " years old." << endl;
    cout << "He weighs " << fido.getWeight() << " lbs." << endl;

    rover.speak();
    fido.speak();

    return 0;
}
```

دو خط اول از برنامه اصلی دو شی fido و rover را از کلاس Dog می سازد (نمونه سازی می کند). سن و وزن Rover را چاپ کرده و سپس با استفاده از توابع دسترسی دهنده setAge و setWeight مقدار آنها را تغییر می دهد. توجه کنید که یک متدها به عنوان بخشی از یک نمونه از کلاس فراخوانی می شوند. آنها بصورت rover.method فراخوانی می شوند. عملگر نقطه "." برای دسترسی به اعضا یا متدهای کلاس استفاده می شود. در ادامه، در برنامه fido مساوی rover مقداردهی می شود. به همین دلیل هر عضو rover در عضوهای fido کپی می شود. همانطور که در درسهای آینده خواهیم دید، این روش کپی اعضا برای همه کلاسها مناسب نیست، مخصوصا وقتی آن کلاس حاوی اشاره گرهایی به اشیای دیگر باشد. در انتها، همانطور که همه سگها انجام می دهند، تصمیم به پارس کردن می گیرند. به عنوان تمرین کد بالا را تایپ و کمپایل کنید. سپس آن را اجرا و نتایج را مشاهده کنید.

شی شناسی

این بخش تعاریفی برای سه لغت اصلی که در طراحی و برنامه نویسی شی گرا با آن روبرو هستید را ارائه می دهد.

کپسوله سازی: کپسوله سازی در حقیقت پنهان سازی داده هاست. داده خصوصی، رابط عمومی. اعضای کلاس خصوصی هستند، متدهای دسترسی عمومی هستند. برنامه های استفاده کننده از کلاس نیازی به دانستن جزییات داخلی کلاس ندارند. اگر محتویات داخلی کلاس تغییر کرد اما رابط به همان شکل ثابت بود، برنامه های استفاده کننده از کلاس نیازی به تغییر ندارند فقط باید دوباره کمپایل شوند.

وراثت: یک زیرکلاس ممکن است از یک کلاس مشتق شده باشد و متدها و اعضای آن را به ارث ببرد. زیر کلاس اختصاصی تر خواهد بود. برای مثال، ما یک کلاس "وسیله نقلیه" که متدها و اعضایی که همه وسایل نقیله لازم دارند را دارد. برای نمونه، همه وسایل نقلیه یک عضو برای ذخیره سرعت و یک تابع برای ترمز دارند. یک کلاس هواپیما از کلاس

وسایل نقلیه مشتق شده و امکانات خاص مانند عضوی به نام "ارتفاع" یا متد "فرود" به آن اضافه می شود. وراثت اجازه توسعه کد بدون اختراع دوباره چرخ را می دهد!

چند ریختی: چند ریختی به یک شی توانایی داشتن رفتارهای متفاوت در موقعیتهای متفاوت بر حسب محتوا را می دهد. برای مثال کلاس عمومی "Car" و کلاسهای مشتق شده "Ford" و "Chevy" ممک است توابع مربوط به شتاب داشته باشند. برنامه استفاده کننده از این کلاسها ممکن است در زمان اجرا یکی از این کلاسها را انتخاب کند و متد شتاب صحیح را به عنوان برنامه اجرا شده انتخاب کند. شکل دیگری از چند ریختی بصورت overloading توابع است. چندین تابع می توانند یک نام داشته باشند اما با آرگومانهای متفاوت. تابع درست با توجه به آرگومانهای استفاده شده انتخاب می شود.

ترجمه: محمد صافدل
منبع: http://cplus.about.com