برنامه نویسی شی گرا

Object Oriented Programming

مروری بر برنامه نویسی ساخت یافته

در دهه 1960 میلادی تولید و توسعه بسیاری از نرم افزارها با مشکل مواجه شد. در آن زمان سبک و ساختار خاصی برای برنامه نویسی وجود نداشت. وجود دستور پرش (goto) نیز مشکلات بسیاری را برای فهم و درک برنامه توسط افراد دیگر ایجاد می کرد، چرا که جریان اجرای برنامه مرتبا دچار تغییر جهت شده و دنبال کردن آن دشوار می گردید.

لذا فرایند تولید نرم افزارها بسیار زمان بر و پرهزینه شده بود و معمولا اشکال زدایی، اعمال تغییرات و گسترش نرم افزارها بسیار مشکل بود. نرم افزارهای تولیدی نیز از قابلیت اعتماد بالایی برخوردار نبود.

تحقیقاتی که برای برطرف کردن این مشکلات به عمل آمد منجر به سبک برنامه نویسی ساخت یافته شد.

مروری بر برنامه نویسی ساخت یافته

برنامه نویسی ساخت یافته روش منظمی برای نوشتن برنامه ها است و منجر به نوشتن برنامه هایی می شود که خوانایی آن ها بالا است و تست و اشکال زدایی آن ها راحت تر و اصلاح آن ها آسان تر است.

ساختار برنامه در این سبک برنامه نویسی به صورت زیر است:

- **تقسیم مساله به مساله هایی کوچکتر**
- **آماده سازی رویه های مناسب برای حل این زیر مساله ها**
- **نوشتن دستورات کامپیوتری مناسب برای این رویه ها**

مروری بر برنامه نویسی ساخت یافته

در برنامه نویسی ساخت یافته می توان هر برنامه ای را بدون دستور پرش و فقط با استفاده از سه ساختار کنترلی **ترتیب**، **انتخاب** و **تکرار** نوشت.

ساختار **ترتیب**، همان اجرای دستورات به صورت متوالی (یکی پس از دیگری) است که کلیه زبانهای برنامه نویسی در حالت عادی به همان صورت عمل می کنند.

ساختار **انتخاب** به برنامه نویس اجازه می دهد که براساس درستی یا نادرستی یک شرط، تصمیم بگیرد کدام مجموعه از دستورات اجرا شود.

ساختار **تکرار** نیز به برنامه نویسان اجازه می دهد مجموعه خاصی از دستورات را تا زمانی که شرط خاصی برقرار باشد، تکرار نماید.

برنامه نویسی شی گرا

برنامه نويسي شئ گرا يا oop يك روش جديد برنامه نويسي مي‌باشد كه در آن علاوه بر ويژگی ساخت يافته بودن برنامه از چند ويژگي قوي جديد استفاده مي‌شود.
برنامه نویسی شئ گرا شیوه نوینی است که قطعات نرم افزاری را ایجاد می کند که در برنامه های مختلف مورد استفاده قرار می گیرند. همانطور که کامپیوتر از قطعات سخت افزاری ساخته می شود در برنامه نویسی شئ گرا برنامه از قطعات نرم افزاری ساخته می شود. به این ترتیب سرعت تولید نرم افزار افزایش می یابد. قابلیت خوانایی برنامه هایی که در این روش نوشته می شوند بالا بوده تست عیب یابی و اصلاح آنها آسان است.

ایده اصلی برنامه نویسی شی گرا

در دنیای اطراف به هر جا نگاه کنید؛ اشیایی را می بینید.

انسان بر اساس اشیا فکر می کند.

در برنامه نویسی شی گرا از توانایی عجیب انسان در انتزاع(تجرید) برای مدل سازی اشیای دنیای واقعی در مفاهیم موجود در زبان برنامه نویسی استفاده می شود.

(در صورت لزوم می توانیم به جای اینکه به درخت ها فکر کنیم؛ به جنگل فکر کنیم)

خواص اساسی روش شی گرا

تجرید(مجرد سازی, انتزاع)

کپسوله کردن یا مخفی سازی اطلاعات

وراثت

چند شکلی

ارتباطات پیامی

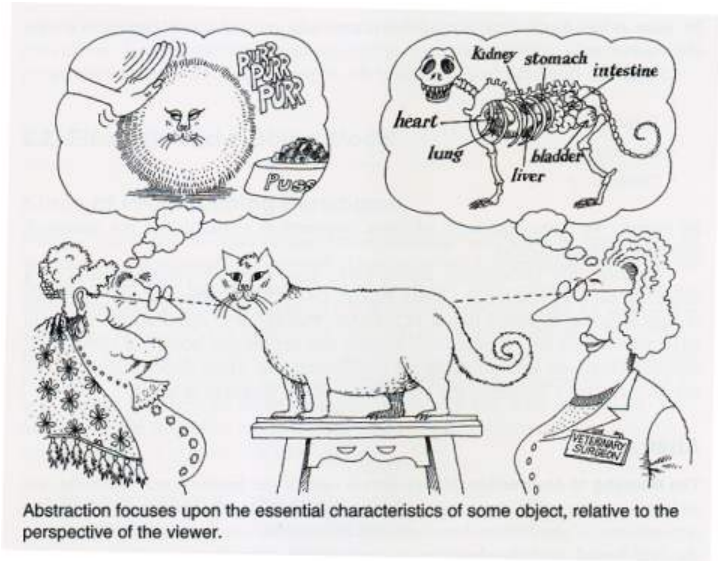
همروندی

قابلیت استفاده مجدد

خواص اساسی روش شی گرا

مجرد سازی

تجرید(مجرد سازی) اصل نادیده گرفتن جنبه هایی از حوزه مسئله است که به هدف الان ما مربوط نیستند . این اصل همچنین به معنای خلاصه سازی می باشد یعنی آنکه ما می توانیم به مسئله از یک دید کلی، به راحتی و بدون لحاظ کردن جزئیات، نگاه کنیم. مثل نقشه کشور، شهر و منطقه.



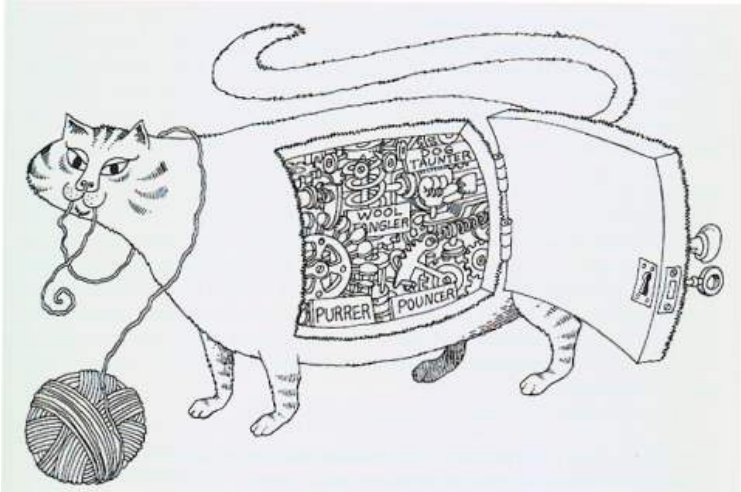
Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

استخراج مدلي از مسئله با در نظر گرفتن مهمترین نکات و حذف جزئیات زائد

خواص اساسی روش شی گرا

محصورسازی

منظور از محصورسازی این است که جزئیات یک فرایند یا عمل از دید استفاده کننده آن مخفی باشد . همچنین صفات و اطلاعات یک شیء از دید سایر اشیاء و اجزاء مخفی باشد و ارتباط از طریق ارسال پیام صورت گیرد .

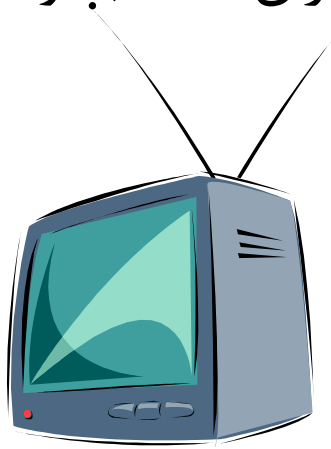


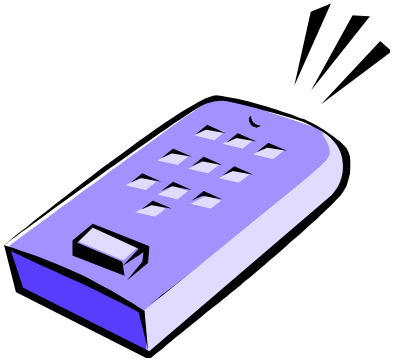
Encapsulation hides the details of the implementation of an object.

خواص اساسی روش شی گرا

محصورسازی

کاربران فقط به واسط استفاده وابسته هستند و نه به جزئیات پیاده‌سازي





خواص اساسی روش شی گرا

وراثت

وراثت، روشی برای بیان شباهت ها است. به عنوان مثال" برای مدل سازی انسان های یک دانشگاه می توانیم آنها را به اشیاء" دانشجو، استاد و کارمند" تفکیک کنیم. اما برخی خصوصیات این اشیاء، مشابه یکدیگر می باشند، نظیر:

" نام، نام خانوادگی، تلفن و. ."

برای اجتناب از تکرار خصوصیات مشترک اشیاء کلاسی به نام " انسان" ایجاد می کنیم که صفات آن همان صفات تکراری سه کلاس اصلی دانشگاه است. سپس هر یک از آن سه کلاس، تمام خصوصیات این کلاس جدید را به ارث می گیرند.

خواص اساسی روش شی گرا

چند شکلی

چند شکلی یا چند ریختی، به معنای یک چیز بودن و چند شکل داشتن است. مثل آب که دارای چند شکلِ جامد، مایع و گاز ظاهر می شود.

خواص اساسی روش شی گرا

ارتباطات پیامی

ارتباطات پیامی راهی است که اشیاء در یک متدولوژی شیءگرا با یکدیگر ارتباط برقرار می کنند. شبیه رویه ها و توابع در زبان های برنامه نویسی که با ارسال پارامتر از نقطه ای درون برنامه، یک رویه یا تابع فراخوانی می شود.

خواص اساسی روش شی گرا

همروندی

همروندی، اجرای هم زمان دو یا چند فعالیت سیستم است. برای مثال در یک چاپگر، می توانیم هم زمان با چاپ نامه مورد نظرمان، نشانه(آرم) شرکت را نیز به عنوان زمینه نامه و هم زمان با متن نامه به چاپ برسانیم.

خواص اساسی روش شی گرا

قابلیت استفاده مجدد

استفاده مجدد قابلیتی است که بیان گر استفاده دوباره از چیزی است که هم اکنون وجود دارد.قابلیت استفاده مجدد خاصیتی است که هر روز از آن استفاده می کنیم مانند کپی کردن اطلاعات و در اختیار دیگران قرار دادن.

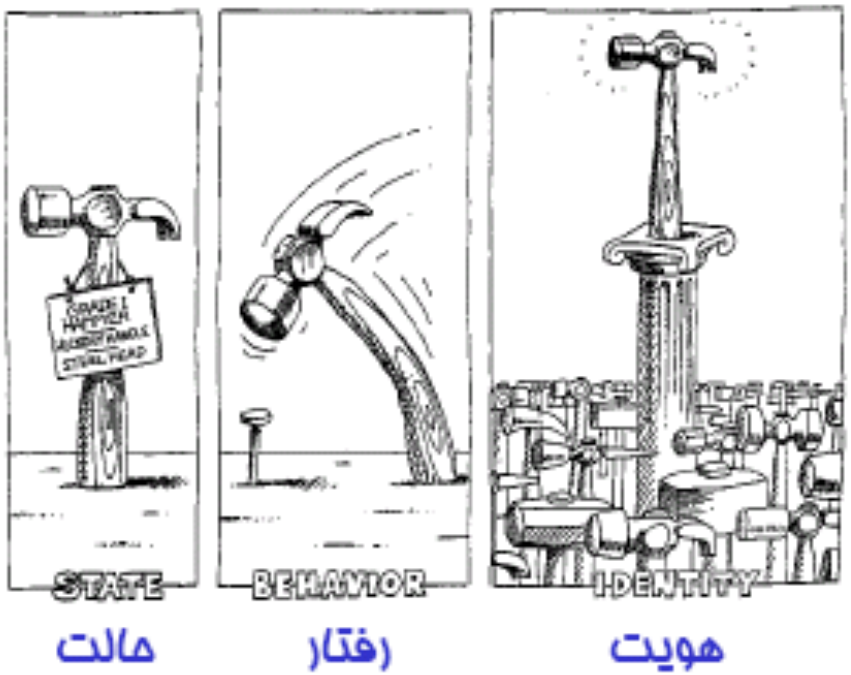
مفاهيم پايه شی گرايي

- شی(Object)
- صفت (Property)
- روش (Method)
- کلاس (Class)

مفاهيم پايه شی گرايي

شی(Object)

شی یک موجودیت فیزیکی یا مفهوم کلی است بگونه ای که دارای هویت بوده و قادر به بروز رفتار و ثبت حالات (وضعیت) خود می باشد.



فرآیند شیمیایی

مفاهيم پايه شی گرايي

شی(Object)

هویت (Identity): آن ویژگی از یک شیء است که آن را از بقیه اشیاء متمایز می سازد.

حالت (State) : وضعيتي كه هر لحظه شئ در آن قرار دارد و با مقدار صفات آن در آن لحظه مشخص مي‌شود.

رفتار(Behavior) : چگونگی عمل و عکس العمل یک شیء در قالب تغییر حالت در مقابل دریافت و یا ارسال پیام را نمایش می دهد.

مفاهيم پايه شی گرايي

شی(Object)

اشياء به دو دسته تقسيم بندی می گردند:

فعال (Active)

غير فعال (Passive)

شیء فعال با فرستادن پيام بر روی اشياء ديگر تاثير می گذارد و شیء غير فعال تاثير می پذيرد.

چنانچه شی هم فعال و هم غيرفعال باشد ، عامل(Agent) ناميده می گردد.

مجموعه‌اي از داده‌ها + اعمال بر روي آن داده‌ها = شیء

مفاهیم پایه شی گرایی

صفت(Attribute)

هر شی یکسری خصوصیات دارد که به آنها صفت گفته می شود که در واقع یک مقدار یا ارزش مشخصی برای آن به ازای هر شی می تواند وجود داشته باشد.

وزن
قد
ارتفاع
طول
عرض
.....

مفاهیم پایه شی گرایی

روش(Method)

هر شی یکسری رفتار دارد که به آنها روش(متد) گفته می شود . متد در واقع پاسخ هایی است که آن شی در مقابل تحریکات محیط از خود نشان می دهد.

مفاهیم پایه شی گرایی

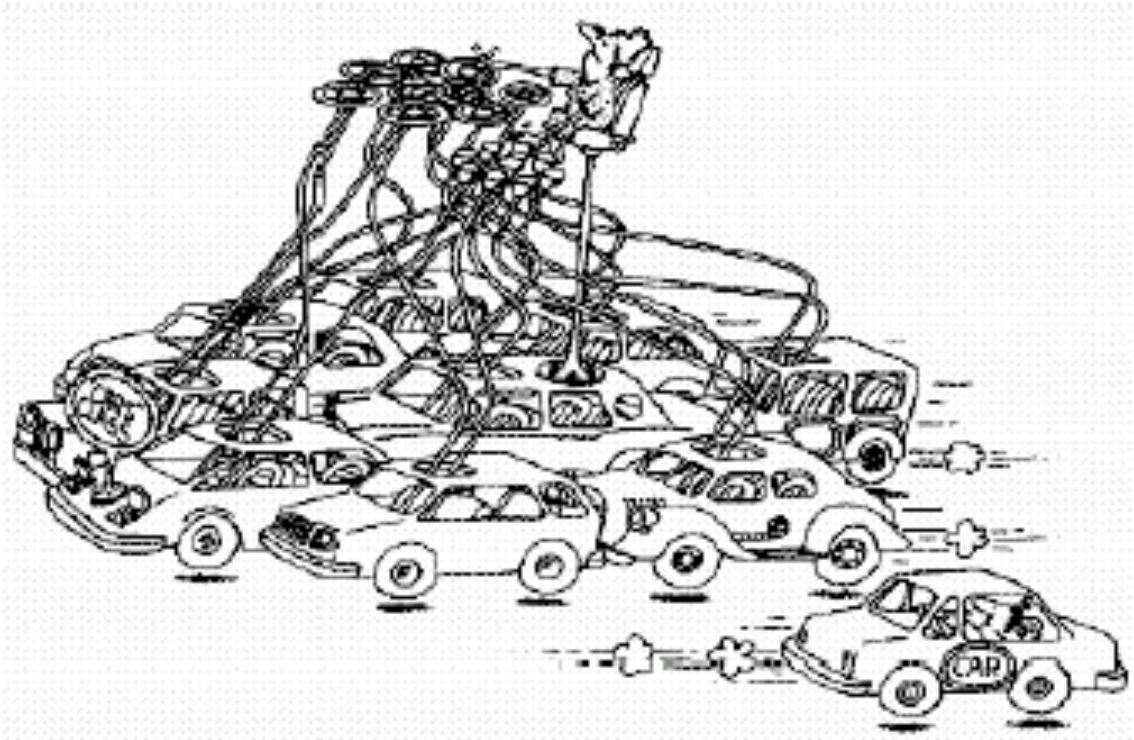
کلاس (Class)

به مجموعه ای از اشیا که دارای ویژگی و رفتار (متد) مشترک می باشند، کلاس گویند.

کلاس ماشین

کلاس انسان

کلاس دانشجو



برنامه نویسی شی گرا

در زبان های برنامه نویسي شی گرا مانند C++؛ اشیاء اساسا از موارد زیر تشکیل می شوند:

- **صفات** :اطلاعاتی که ویژگی های یک شی را بیان مي‌کنند. این اطلاعات ممکن است ساده مانند اعداد صحیح، رشته ها و اعداد حقیقی ویا ممکن است ارتباط به شيء پیچیده داشته باشد.
- **روش‌ها**: روش ها رفتار یک شی را تعریف مي کنند و در حقیقت همان رویه ها و توابع در برنامه نویسي هستند.

تمرکز برنامه‌نویسان $C++$ بر روی ایجاد «نوع‌های تعریف شده از سوی کاربر» که کلاس نامیده می‌شوند است. هر کلاس حاوی داده و هم مجموعه‌ای از متدها است که بر روی داده‌ها کار می‌کنند و سرویس‌هایی برای سرویس‌گیرنده‌ها (کلاس‌ها و توابع دیگری که از کلاس استفاده می‌کنند) تدارك می‌بیند

برنامه نویسی شی گرا

کلاس‌ها همانند نقشه ترسیمی خانه‌ها هستند. یك کلاس، نقشه ایجاد یك شی از کلاس است. همان طوری که می‌توانیم خانه‌های متعددی از روی یك نقشه بسازیم، می‌توانیم تعدادی شی از روی یك کلاس، نمونه‌سازی کنیم. نمی‌توان در نقشه آشپزخانه مبادرت به آشپزی کرد، آشپزی فقط در آشپزخانه خانه امکان‌پذیر است.

مدل شی گرا

یک مدل شی گرا مجموعه ای است از اشیاء و کلاس ها که در جهت پیاده سازی رفتار کل سیستم به یکدیگر پیغام می فرستند و اعمالی را انجام می دهند. یک شی، ساختمان داده و رفتار مربوطه اش را یک جا و به طور مجتمع در خود نگاه می دارد.

تعریف کلاس (اعلان کلاس، واسط کلاس)

class نام کلاس {

داده ها و توابع اختصاصی

public:

داده ها و توابع عمومی

private:

داده ها و توابع اختصاصی

protected:

داده ها و توابع محافظت شده

اشیای کلاس;}

داده های کلاس را عضو داده ای و متدهای کلاس را تابع عضو می نامیم

کنترل های دسترسی به اعضای کلاس

کلمات public، private، protectedتعيين كنندة نحوه دسترسي استفاده كنندگان از كلاس به ويژگيها و رفتارهای (متدها) كلاس مي‌باشند.

public:

به معني دسترسي براي عموم مي‌باشد.

private:

به معني دسترسي فقط براي اعضاي كلاس مي‌باشد.

protected:

به معني دسترسي براي اعضاي كلاس و ارث‌برندگان كلاس مي‌باشد.

نکته: داده ها و توابعی که بلافاصله بعد از نام کلاس می آیند مختص کلاس هستند (همانند private).

تعریف توابع عضو کلاس (پیاده سازی)

```
(آرگومان های ورودی) نام تابع :: نام کلاس    نوع خروجی تابع
{
...
}
```

مثال: تعریف کلاس و پیاده سازی

```
Class CPoint {
   int x, y;
public:
   double z, t;
   void PrintValues();
private
   float s, d;
};


Void CPoint :: PrintValues()
{
 cout << "x=" << x << ",  y="  << y << endl;
}
```

نمونه سازی کلاس (تعریف شی)

مدلسازی اشیای دنیای واقعی در C++:

قدم اول : تعریف کلاس است.

(تعریف کلاس، اشیای واقعی ایجاد نمی کند.)

قدم دوم: نمونه سازی کلاس(Instantiation).

(نمونه سازی، شی ای از نوع کلاس ایجاد می کند.)

اعلان شی از نوع کلاس:

نام کلاس نام شی;

CPoint start, end, X, Y, Z;

دستیابی به اعضای یک شیء

پس از ایجاد یک شیء، می توانیم به اعضای آن دستیابی داشته باشیم. برای دستیابی به اعضای یک شیء به صورت زیر عمل می کنیم:

نام شیء . نام عضو(داده ای یا تابعی);

```
start.z;
start.PrintValues();
```

نکته: در خارج از یک شیء، فقط می توان به اعضای عمومی(public) آن دست یافت.

start.x 

دستیابی به اعضای اختصاصی یک شیء

برای دستیابی به یک عضو(فیلد) اختصاصی یک شی در خارج از آن شیء، باید تابع عضوی(متدی) بنویسیم که مقدار آن فیلد را به ما بدهد. این متد را متد بازیابی(accessor)نامند

این متد معمولا به صورت getAttribute() نامگذار ی می شود که به جای Attribute**، نام صفت (فیلد) فرار می گیرد.**

مثلا برای دسترسی به عضو x از شیء start از متدی به نام getX() استفاده می کنیم:

```
int getX()
{
  return x;
}
```

دستیابی به اعضای اختصاصی یک شیء

```
Class CPoint {
   int x، y؛
public:
   double z، t؛
   int getX()؛

private
   float s، d؛
}؛

int CPoint :: getX()
{
 return x؛
}
```

تغییر مقدار اعضای اختصاصی یک شیء

برای تغییر مقدار یک عضو(فیلد) اختصاصی یک شی از متدهای تغییر دهنده mutator **استفاده می کنیم.**

این متد معمولا به صورت setAttribute() **نامگذار ی می شود که به جای** Attribute**، نام صفت (فیلد) فرار می گیرد.**

مثلا برای تغییر مقدار عضو x از شیء start از متدی به نام ( )setX استفاده می کنیم. که در آن a به عنوان مقدار جدید عضو داده ای x در نظر گرفته می شود.

```
void setX(int a)
{
   x = a;
}
```

تغییر مقدار اعضای اختصاصی یک شیء

```
Class CPoint {
   int x، y؛
public:
   double z، t؛
   void SetX(int a)؛

private
   float s، d؛
}؛

Void CPoint :: SetX(int a)
{
 x=a؛
}
```

سازنده کلاس

در C++ متدي از کلاس که همنام آن کلاس است سازنده کلاس نامیده می شود. به این معني که موقع ایجاد شی ای از نوع آن کلاس، ابتدا متد (تابع عضو) مذکور اجرا می شود.

سازنده کلاس، هیچ مقداری برنمی گرداند(حتی void**).**

سازنده کلاس، با کنترل دستیابی public **مشخص می شود.**

سازنده کلاس، به طور خودکار هنگام ایجاد شیء ای از کلاس اجرا می شود و صریحا فراخوانی نمی شود.

اگر در تعریف کلاس، صریحا سازنده ای تعریف نشود، هنگام تعریف شیءای از کلاس، به انواع داده ای عددی مقدار پیش فرض صفر و به مقادیر رشته ای مقدار تهی نسبت داده می شود.(سازنده پیش فرض)

نکته: اگر عضوی از کلاس، شیءای از کلاس دیگر باشد، سازنده ی آن شیء برای مقدار اولیه دادن به این شیء اجرا می شود.

سازنده کلاس (سازنده بدون پارامتر)

```
Class CPoint {
   int x, y;
public:
   double z, t;
   CPoint();
   void PrintValues();
private
   float s, d;
};

CPoint :: CPoint()
{
 x=10;
 y=5;
}

Void CPoint :: PrintValues()
{
 cout << "x=" << x << ",  y="  << y << endl;
}
```

نکته: تابع**سازنده بدون پارامتر، به طور خودکار هنگام ایجاد شیءای از کلاس اجرا شده و مقادیر پیش فرضی را به اعضای داده ای شیء نسبت می دهد.**

سازنده کلاس (سازنده بدون پارامتر)

```
CPoint start;
start.PrintValues();
```

خروجی قطعه برنامه فوق به صورت زیر است:

```
x=10, y=5
```

سازنده کلاس (سازنده با پارامتر)

نکته: تابع**سازنده کلاس می تواند با پارامتر باشد. معمولا این پارامترها برای مقدار اولیه دادن به اعضای شیء به کار می روند.** تابع**سازنده به طور خودکار هنگام ایجاد شیء ای از کلاس اجرا شده و آرگومانهای خودش را به اعضای داده ای شیء نسبت می دهد.**

```
Class CPoint {
   int x، y؛
public:
   double z، t؛
   CPoint(int a، int b)؛

private
   float s، d؛
}؛

CPoint :: CPoint(int a، int b)
{
 x=a؛
 y=b؛
}
```

سازنده کلاس (سازنده با پارامتر)

```
CPoint start(2,5);
CPoint end=CPoint(6,7);
```

نکته: این دو شیوه ی ایجاد و مقداردهی اولیه به اشیاء،دقیقا یکسان می باشند و هیچ تفاوتی با هم ندارند.

```
start.PrintValues();
end.PrintValues();
```

خروجی قطعه برنامه فوق به صورت زیر است:

```
x=2, y=5
x=6, y=7
```

سازنده کلاس (سازنده ی دارای آرگومان های پیش فرض)

نکته: تابع**سازنده کلاس می تواند دارای آرگومان های پیش فرض باشد. با تعیین آرگومان های پیش فرض برای سازنده، حتی اگر هیچ مقداری در فراخوانی سازنده تدارک دیده نشود، ارزش دهی اولیه اشیاء انجام می شود.**

```
Class CPoint {
   int x, y;
public:
   double z, t;
   CPoint(int =34, int =56);

private
   float s, d;
};

CPoint :: CPoint(int a, int b)
{
 x=a;
 y=b;
}
```

سازنده کلاس (سازنده ی دارای آرگومان های پیش فرض)

```
CPoint start=CPoint(); //or  CPoint start;
CPoint end=CPoint() ; //or  CPoint end;

start.PrintValues();
end.PrintValues();
```

خروجی قطعه برنامه فوق به صورت زیر است:

```
x=34, y=56
x=34, y=56
```

نکته: در اولین دستور، به عضو داده ای x از شیء start مقدار ۳۴ و به عضو داده ای y از شیء start مقدار ۵۶ نسبت داده می شود.

در دومین دستور، به عضو داده ای x از شیء end مقدار ۳۴ و به عضو داده ای y از شیء end مقدار ۵۶ نسبت داده می شود.

سازنده کلاس (سازنده ی دارای آرگومان های پیش فرض)

```
CPoint start=CPoint(3); //or  CPoint start(3);
CPoint end=CPoint(4); //or  CPoint end(4);


start.PrintValues();
end.PrintValues();
```

خروجی قطعه برنامه فوق به صورت زیر است:

```
x=3, y=56
x=4, y=56
```

نکته: در اولین دستور، به عضو داده ای x از شیء start مقدار ۳ و به عضو داده ای y از شیء start مقدار 56نسبت داده می شود.

در دومین دستور، به عضو داده ای x از شیء end مقدار ۴ و به عضو داده ای y از شیء end مقدار 56نسبت داده می شود.

سازنده کلاس (سازنده ی دارای آرگومان های پیش فرض)

```
CPoint start=CPoint(3،4)؛ //or  CPoint start(3،4)؛
CPoint end=CPoint(5،6) ؛ //or  CPoint end(5،6)؛


start.PrintValues()؛
end.PrintValues()؛
```

خروجی قطعه برنامه فوق به صورت زیر است:

```
x=3، y=4
x=5، y=6
```

نکته: در اولین دستور، به عضو داده ای x از شیء start مقدار ۳ و به عضو داده ای y از شیء start مقدار ۴ نسبت داده می شود.

در دومین دستور، به عضو داده ای x از شیء end مقدار ۵ و به عضو داده ای y از شیء end مقدار ۶ نسبت داده می شود.

تعریف چند تابع سازنده در یک کلاس

```
Class CPoint {
   int x, y;
public:
   double z, t;
   CPoint();
   CPoint(int a, int b);
private
   float s, d;
};

CPoint :: CPoint()
{
 x=8;
 y=9;
}

CPoint :: CPoint(int a, int b)
{
 x=a;
 y=b;
}
```

تعریف چند تابع سازنده در یک کلاس

```
CPoint star=CPoint();   //or  CPoint start;
CPoint end=CPoint(5,6); //or  CPoint end(5,6);


start.PrintValues();
end.PrintValues();
```

خروجی قطعه برنامه فوق به صورت زیر است:

```
x=8, y=9
x=5, y=6
```

نکته:

در اولین دستور، به عضو داده ای x از شیء start مقدار پیش فرض ۸ و به عضو داده ای y از شیء start مقدار پیش فرض ۹ نسبت داده می شود.

در دومین دستور، به عضو داده ای x از شیء endمقدار ۵ و به عضو داده ای y از شیء end مقدار ۶ نسبت داده می شود.

مخرب کلاس

در C++ متدي از کلاس که همنام آن کلاس و با یک علامت ~ قبل از آن است مخرب کلاس است. به این معني که موقع از بین رفتن شئ آن کلاس، متد مذکور اجرا مي‌شود.

مخرب کلاس، هیچ مقداری برنمی گرداند(**حتی** void).

مخرب کلاس، با کنترل دستیابی public **مشخص می شود.**

مخرب کلاس، به طور خودکار هنگام خاتمه برنامه اجرا می شود.

اگر در تعریف کلاس، صریحا مخربی تعریف نشود، کامپایلر، یک مخرب خالی ایجاد می کند.

خود مخرب حافظه ی شیء را از بین نمی برد، بلکه زمینه را برای حذف حافظه ی شیء آماده می کند.

مخرب کلاس

```
Class CPoint {
   int x, y;
public:
   double z, t;
   CPoint(int =34, int =56);
  ~CPoint();

private
   float s, d;
};

CPoint :: CPoint(int a, int b)
{
 x=a;
 y=b;
}

CPoint :: ~Cpoint()
{
 cout << "DESTRUCTco۲ CPoint" << endl;  //e.g  free(pointer)
 cin.get();
}
```