گونه داده های انتزاعی وکلاس
Abstract Data Types (ADTs) and Class

بهزاد منتظری

دانشکده فنی ومهندسی دانشگاه رازی

نیمسال اول ۸۴-۸۳

گونه داده های انتزاعی

- یک گونه داده انتزاعی از مجموعه ای ازصفات و مجموعه ای از عملیات تشکیل شده که روی این صفات عمل میکنند:

Real Number: <R , {+,-,*,/}>

Boolean: <{F, T} , {and, or, not}>

Stack: < N, {pop, top, push, size, makeEmpty}

- معرفی گونه داده های انتزاعی در زبان های شیئی گرا (برخلاف زبان های پردازه ای) بسیار سرراست میباشد. با دستورclass

Procedural Programming

- Verb Oriented
  - تجزیه بر اساس افعال (*Operations*)
  - تجزیه عملیات به یک سری از عملیات ساده تر
- زبان های برنامه سازی:
  - C
  - Pascal

Procedural Programming

```
Data1
Data2                    Nouns
Data3


Operation1 {
    ----------
    ----------
}                        Verbs
Operation2 {
    ----------
    ----------
}
```

معایب برنامه سازی پردازه ای

- دست کم گرفتن داده ها – به عملیات اهمیت بیشتری نسبت به داده ها داده شده
- درسیستم هایی که بر اساس عملیات تجریه شده اند، مرتبط کردن عناصرسیستم با دنیای واقعی مشکل است. در دنیای واقعی عملیات وجود ندارند
- ایجاد گونه های جدید داده ها مشکل است- قابلیت بسط سیستم کاهش میابد

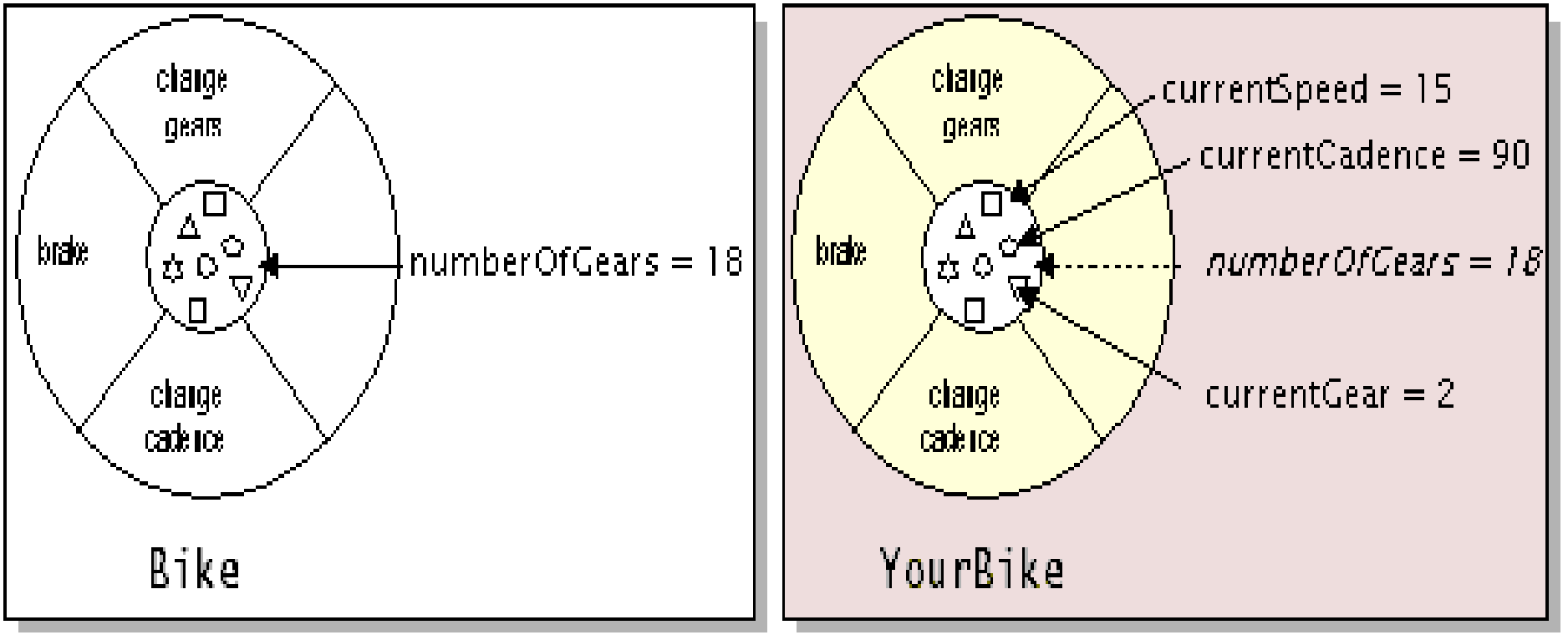
Object Oriented Programming

- شیئی چیست؟
- اشیاء دنیای واقعی چیز هایی هستند که دارای اجزاء زیر میباشند:
  - وضعیت State
  - رفتار Behavior یا عملیات (operation)
- مثلا "این کامپیوتر"
  - وضعیت : اندازه RAM، نوع پردازنده
  - عملیات: Reset, On/Off
- درنرم افزار یک شیئ بسته ای از متغیر ها (وضعیت) و متد (operation) ها است

کلاس چیست؟

- کلاس طرحی است که در آن متغیر ها و متد های رایج در همه اشیاء یک گونه معرفی میشوند.
- مثلا "این کامپیوتر" یک شیئ متعلق به کلاس "کامپیوتر" است
- یک شیئ مقادیر متغیر های معرفی شده در کلاس را نگهداری میکند (مثلا نوع پردازنده = Pentium 4)
- به شیئ یک عضو (Instance) از کلاس نیزگفته میشود

مقایسه کلاس و شیئ

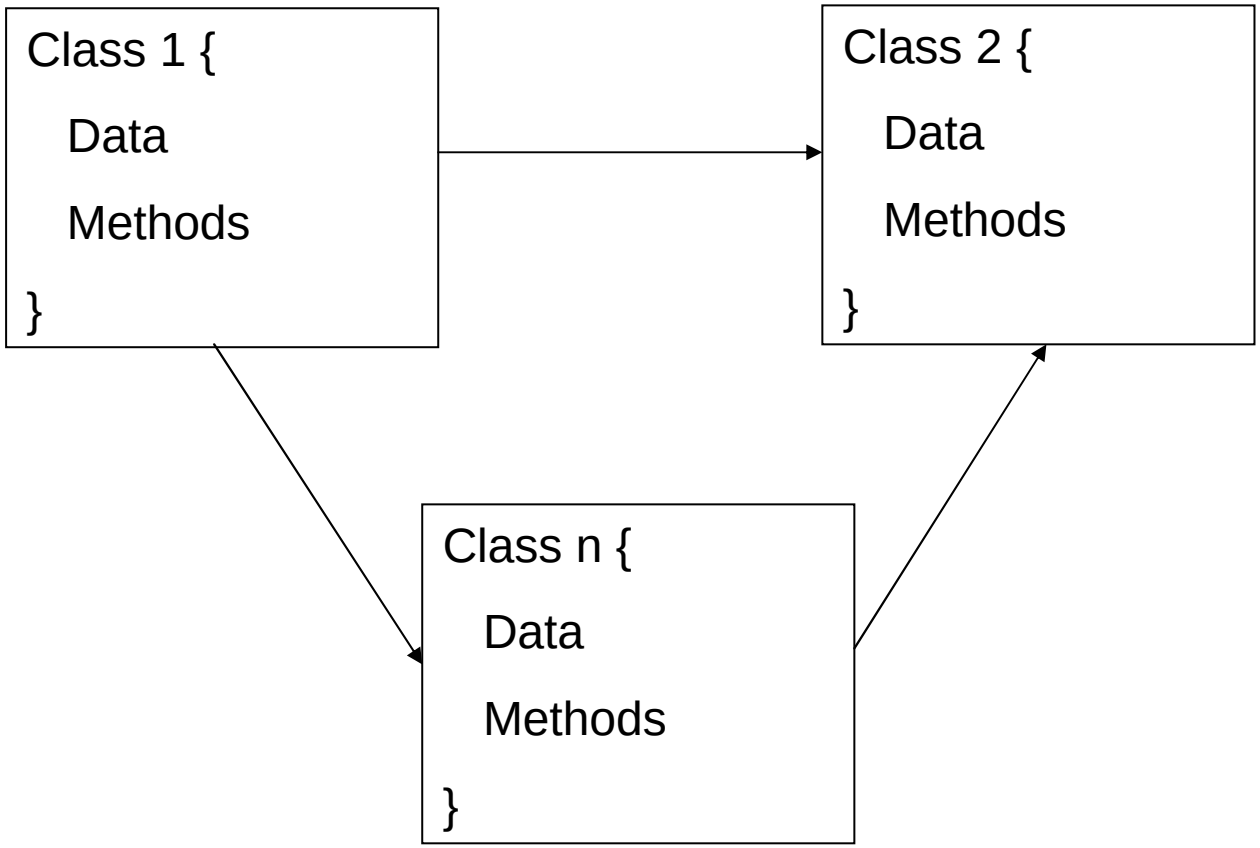


- به NumberOfGears متغیر کلاسی *Class Variable* گوییم. این متغیر یک مقدار را برای همه اشیاء کلاس در خود نگهداری میکند.

برنامه سازی شیئی گرا

- بر حسب اشیاء (Nouns) و روابط بین آنها بیان میشود
- زبان های برنامه سازی
  - C++ –
  - Java –
  - C# –

برنامه سازی شیئی گرا

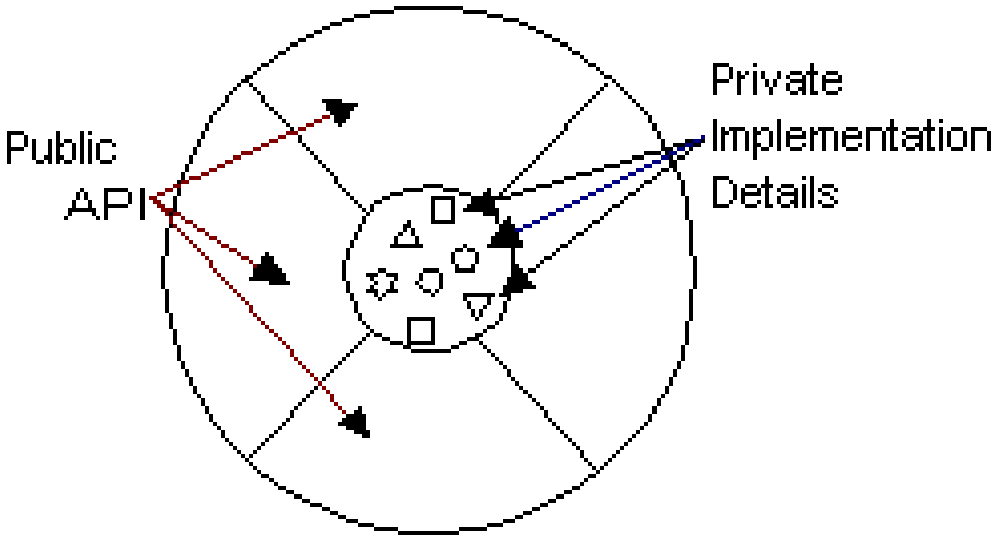


(Object Oriented) OO اصول

- کپسوله سازی Encapsulation
- توارث Inheritance
- چند ریختی Polymorphism

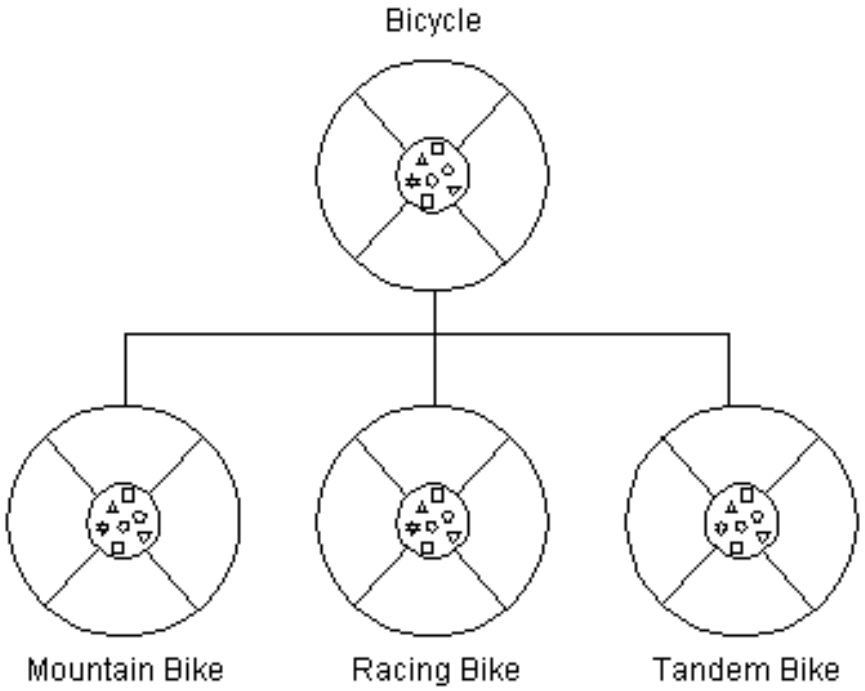
Encapsulation

- حصار کشی داده ها در یک شیئ
- داده ها از بیرون مستقیما قابل دستیابی نیستند
- ارائه امنیت داده ها



Inheritance

- یک کلاس *(SubClass)* میتواند صفات و متد های کلاس دیگری (SuperClass) را به ارث ببرد
- زیر کلاس رفتار تخصصی specialized را ارائه میکند
- امکان استفاده مجدد از کد *Code Reuse* را بنحو مناسب فراهم میکند
- جلوگیری از افزونگی داده ها



Polymorphism

- قابلیت گرفتن فرم های مختلف
- متدی که در کلاس پایه معرفی شده در زیر کلاس ها رونویسی overridden میشود تا رفتار مناسب در زیر کلاس را ارائه کند
- مثلا کلاس پایه "چند وجهی" دارای متد getArea است

  متد getArea در زیر کلاس "مثلث" -> a=x*y/2

  متد getArea در زیر کلاس "مستطیل" -> a=x*y

کلاس در C++

- مثال اعداد مختلط
- صفات و عملیات زیررا برای Complex Number ADT در نظر میگیریم:

Attributes: real, imaginary

Behaviour: Make_Complex_Number()
Set_Real_Part()
Set_Imaginary_Part()
Get_Real_Part()
Get_Imaginary_Part()
+, -, *, /
Print()

کلاس در C++

- گونه اعداد مختلط باید بنحوی معرفی شود که بتوان ازآن بصورت زیر استفاده کرد(شبیه گونه های درونی) :

```
void main(){
Complex_Number c1(2,3),b,c;
b.Set_Real_Part(-1);
b.Set_Imaginary_Part(2);
c=a+b; //or c= a.operator+(b);
c.print(cout);
cout<<c;
}
```

معرفی کلاس در C++

```
class Complex_Number
{
    private:
        double real;
        double imaginary;
    public:
        void Set_Real_Part(double);
        void Set_Imaginary_Part(double);
        double Get_Real_Part();
        double Get_Imaginary_Part();
        Complex_Number operator+(Complex_Number);
        …
};
```

اظهار کلاس در C++

- فرم کلی اظهار کلاس:

```
class <class name>
{
    private:
            < private members >
     public:
            < public members >
};
```

تعریف متد های کلاس

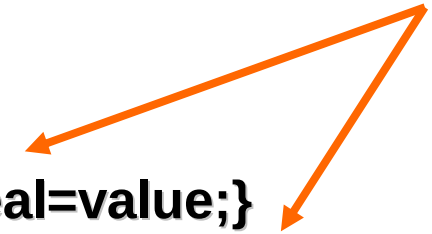
- معرفی رفتار کلاس در زبان C++ در دو محل امکان پذیر است:

۱. در داخل اظهار کلاس
۲. در خارج اظهار کلاس

معرفی متد در داخل کلاس

```
class Complex_Number
{
    private:
        double real;
        double imaginary;
    public:
        void Set_Real_Part(double value){real=value;}
        void Set_Imaginary_Part(double value) {imaginary=value;}
        double Get_Real_Part() {return real;}
        double Get_Imaginary_Part() {return imaginary;}
        Complex_Number operator+(Complex_Number c) {
                Complex_Number h;
                h.real = real + c.real;
                h. imaginary = imaginary + c. imaginary
                return h;
        }
        …
};
```

In-line Functions



معرفی متد در داخل کلاس

شیوه معرفی متد در داخل کلاس معمولا برای متد های ساده بکار میرود

معرفی متد در خارج از کلاس

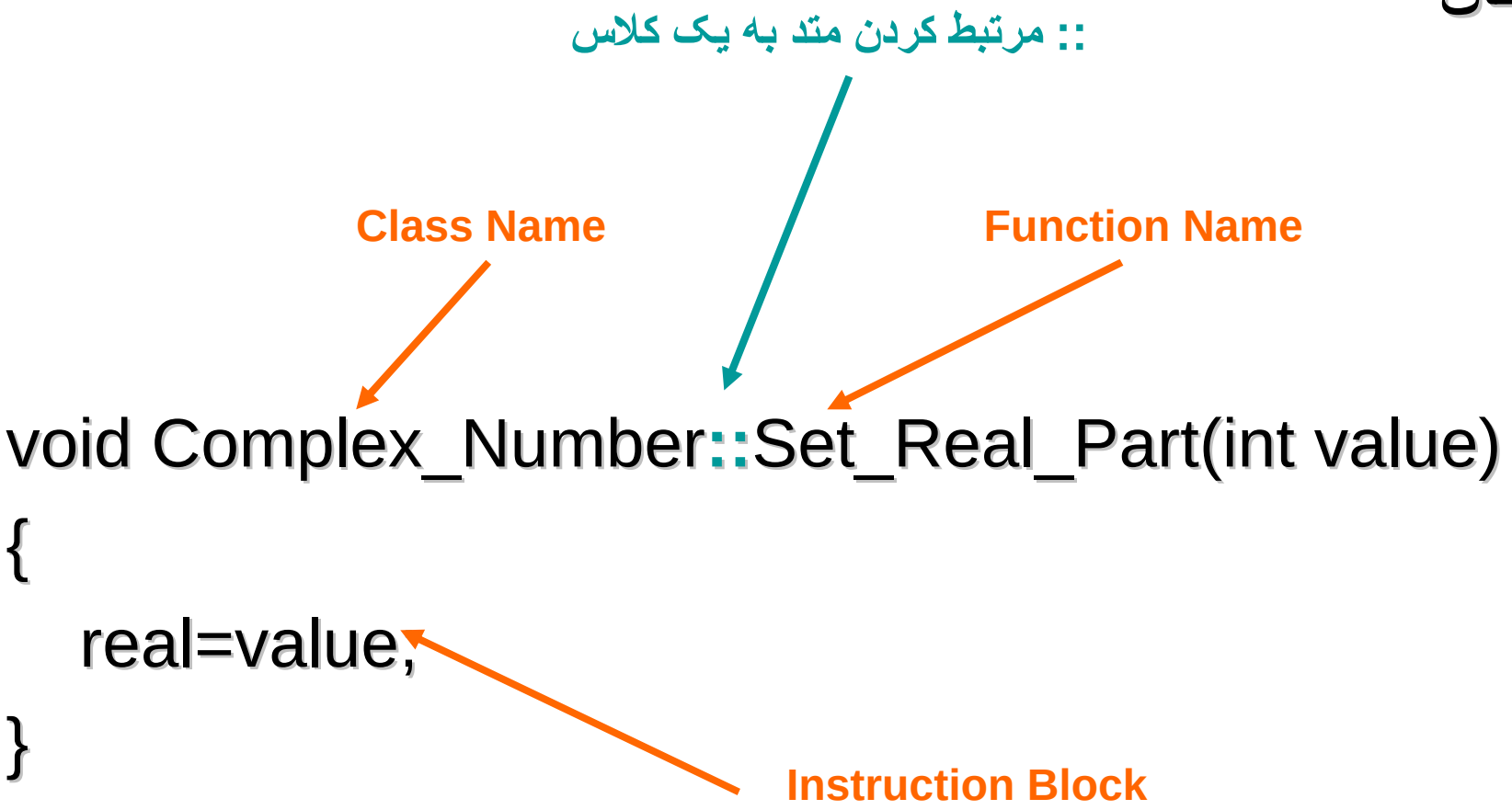
متد ها میتوانند در خارج از کلاس نیز معرفی شوند

(اغلب بلافاصله پس ازتعریف کلاس)

در تعریف متد ها در خارج از کلاس لازم است دقیقامعلوم شود متد متعلق به چه کلاسی است (توجه: به دلیل پلی مرفیسم ممکن است چندین کلاس دارای متد هایی با پروتکل یکسان وجود داشته باشند)

معرفی متد در خارج از کلاس

مثال



Creating Instances

- پس از تعریف کلاس و رفتار آن، میتوان شیئی از کلاس اظهار نمود.
- برای اینکار دو راه وجود دارد:
- روش اول:

```
Complex_Number c;
```

دراین روش به صفات شیئ مقداری تخصیص داده نشده است.
برای اینکار باید از متد های کلاس استفاده کرد:

```
c.Set_Real_Part(-1);
c.Set_Imaginary_Part(2);
```

- برای دستیابی به اجزاء (داده هاومتد) یک شیئ از عملگر(.) Dot Operator استفاده میشود

Class

- توجه : فقط اجزاء عمومی میتوانند با عملگر (.) دستیابی شوند(encapsulation principle)
- مقدار دهی به صفات اشیاء بکمک متد ها (روش بیان شده در اسلاید قبل) در مورد اشیائی که تعداد صفات آنها زیاد است کاری پر زحمت است. راه حل مناسب استفاده از متد(های) سازنده است.

Constructors

- متد سازنده در زمان ایجاد یک شیئ بطور خود کار فرا خوانی میشود.
- وظیفه آن مقدار دهی آغازی به شیئ است
- نام متد سازنده دقیقا با نام کلاس یکی است
- متد سازنده فاقد مقدار بازگشتی است (حتی void)
- یک کلاس میتواند دارای چندین متد سازنده باشد که تفاوت آنها در تعداد و / یا گونه پارامتر ها است.

```
Complex_Number(double, double);
Complex_Number(double );
Complex_Number();
```

Constructors

- متد سازنده نیز همانند سایر متد ها باید تعریف شود(در داخل کلاس یا خارج از آن)

```
Complex_Number::Complex_Number(double re, double im)
{
   real=re;
   imaginary =im;
}
```

روش بهتر- استفاده از گرامر مقدار آغازی دهی به متغیر های شیئی است:

```
Complex_Number::Complex_Number(double re, double im)
   :real(re), imaginary(im){}
```

فراخوانی سازنده

```
Complex_Number c1(2,3); // c1 ← 2+3i
Complex_Number c2(4); // c2 ← 4+ 0i
Complex_Number c3(); // c3 ← 0 + 0i
```

- یک ایجاد پشته بصورت پویا:

```
Complex_Number *c= new Complex_Number(4, 3);
```

معرفی متد Print()

```
void print(ostream o)
{
  o<< real;
  if(imaginary>=0) o << ‘+’;
  o<< imaginary <<‘i’;
}
```

سربارگذاری عملگرهابصورت عمومی خارج ازکلاس

- لزوم: هرگاه عملوند سمت چپ از گونه ای باشد که به کلاس آن دسترسی نداشته باشیم (و یا از گونه های درونی باشد)
- مثال

```
double *   Complex_Number
ostream << Complex_Number
```

- در اینجا عملگر * نمیتواند در کلاس Complex_Number تعریف ویا حتی اظهار شود.

```
double operator *(double left_op, Complex_Number right_op)
```

سربار گذاری عملگر >> برای خروجی اعداد مختلط

- در فایل شمولی <iostream.h> برای خروجی گونه های درونی سربار گذاری های زیر انجام شده است:

```
ostream& operator<<(ostream& o, int i);
ostream& operator<<(ostream& o, double x);
ostream& operator<<(ostream& o, char c);
ostream& operator<<(ostream& o, char* str);
```

به طریق مشابه میتوان برای گونه اعداد مختلط نیز سربار گذاری لازم راانجام داد:

```
ostream& operator<<(ostream& o,Complex_Number &c){
  c.print(o);
  return o;
}
```