3-MA’RUZA.

SINF VA OBYEKT TUSHUNCHALARI. METODLAR, XOSSALAR. SINFLARDA INKAPSULYATSIYA PRINSIPINI QO’LLASH

Sinflar va ularni e‘lon qilish

Sinflar. C++ tilining eng foydali xususiyatlaridan biri bu muayyan muammolarni hal qilish uchun ko‘proq mos keladigan ma'lumotlar turlarini aniqlashdir.

Obyektga yo‘naltirilgan dasturlashda ma'lumotlar turlari nafaqat ma'lumotlarni, balki shu ma'lumotlar bilan ishlaydigan funksiyalarni ham o‘z ichiga olishi mumkin. C++ bu ma'lumot turini aniqlash uchun **class** kalit so‘zidan foydalanadi. **class** kalit so‘zidan foydalanish foydalanuvchi tomonidan belgilanadigan yangi ma'lumotlar turini – sinfni aniqlaydi.

C++ da sinflar strukturalarga juda o‘xshash, faqat ular ko‘proq kuch va moslashuvchanlikni ta'minlaydi. Aslida, quyidagi struktura va sinf funksional jihatdan bir xil:

```
struct DateStruct
{
    int m_day;
    int m_month;
    int m_year;
};
```

```
class DateClass
{
public:
    int m_day;
    int m_month;
    int m_year; };
```

Bu yerda faqat muhim farq - bu **public** kalit so‘zidir.

Strukturani e'lon qilish singari, sinfni e'lon qilish ham hech qanday xotira ajratmaydi. Sinfdan foydalanish uchun siz ushbu turdagi o‘zgaruvchini e'lon qilishingiz kerak:

DateClass today { 3, 8, 2022 };

C++ da sinf o‘zgaruvchisi sinfning nusxasi (yoki "obyekt") deb ataladi. Ma'lumotlar turidagi (masalan, **int x**) o‘zgaruvchini aniqlash, shu o‘zgaruvchiga xotira ajratilishiga olib keladi, shuning uchun sinf obyektini yaratish (masalan, **today DateClass**) bu obyekt uchun xotira ajratilishiga olib keladi.

SINF METODLARI VA XOSSALARI

Ma'lumotlarni saqlashdan tashqari, sinflar funksiyalarni ham o‘z ichiga olishi mumkin. Sinf ichida aniqlangan funksiyalar ***metodlar*** deb ataladi. Metodlarni sinf ichida ham, tashqarisida ham aniqlash mumkin. Hozircha biz ularni sinf ichida belgilaymiz (tushunarli bo’lishi uchun), ularni sinfdan tashqarida qanday aniqlash mumkin haqida keyingi boblarda ko‘rib chiqamiz.

DateClass sinfi tarkibida **print()** metodini e’lon qilish:

```
1  class DateClass
2  {
3  public:
4  int m_day;
5  int m_month;
6  int m_year;
7  void print() // funksiya-a’zoni aniqlash
8  {
9  cout << m_day << "/" << m_month << "/" << m_year;
   }
   };
```

Xuddi shu tarzda, struktura a'zolariga ham, sinf a'zolariga ham (o‘zgaruvchilar va funksiyalarga) a'zo tanlash operatori **(.)** orqali kirish mumkin:

```
#include <iostream>
class DateClass
{
public:
   int m_day;
   int m_month;
   int m_year;
   void print()
   {
      cout << m_day << "/" << m_month << "/" << m_year;
   }
};
```

```
int main()
{
    DateClass today { 3, 8, 2022 };
    today.m_day = 18; // DateClass today obyektining m_day a'zo o‘zgaruvchisini tanlash uchun a'zo tanlash operatoridan foydalanish
    today.print(); // DateClass today obyektining print() metodini chaqirish uchun a'zo tanlash operatoridan foydalanish
    return 0;
}
```

Agar yangi obyekt yaratib, uning nomini tomorrow deb nomlasak, tomorrow obyekti orqali tomorrow.print() ga murojaat qiladigan bo‘lsak, m_day tomorrow.m_dayga ishora qiladi.

Asosan, bog‘langan obyekt bilvosita metodga o‘tkaziladi. Shu sababli, u ko‘pincha yopiq obyekt deb ataladi.

A'zo o‘zgaruvchilar uchun m_ (inglizcha "m" = "members" – a’zolar) prefiksidan foydalanish a'zo o‘zgaruvchilarini funksiya parametrlari yoki sinf metodlari ichidagi lokal o‘zgaruvchilardan ajratishga yordam beradi. Bu bir necha sabablarga ko‘ra foydalidir:

Birinchidan, m_ prefiksli o‘zgaruvchini ko‘rganimizda, biz sinf a'zolari o‘zgaruvchisi bilan ishlayotganimizni tushunamiz;

Ikkinchidan, funksiya parametrlari yoki funksiya ichida e'lon qilingan lokal o‘zgaruvchilardan farqli o‘laroq, a'zo o‘zgaruvchilar sinf ta'rifida e'lon qilinadi.

Odatda dasturchilar sinf nomlarini katta harf bilan yozadilar(1-listing).

```
#include <iostream>
#include <string>
using namespace std;
class Employee
{
public:
    string m_name;
    int m_id;
    double m_wage;
```

```
    // Xodim haqidagi ma'lumotlarni ekranda ko‘rsatish metodi
    void print()
    {
        cout << "Ismi: " << m_name <<endl;
        cout<<" Id: " << m_id <<
        cout<< "Maoshi: " << m_wage << '\n';
    }
};

int main()
{
    // Ikki ishchini aniqlash
    Employee john { "John", 5, 30.00 };
    Employee max { "Max", 6, 32.75 };
   // Xodimlar haqidagi ma'lumotlarni ekranda ko‘rsatish
    john.print();
    cout<<endl;
    max.print();
    return 0;
}
```

SINF XOSSALARI VA METODLARIGA KIRISH SPETSIFIKATORLARI SPETSIFIKATORLARGA KIRISH.

Quyidagi sinfni ko‘rib chiqaylik:

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

//sinfni e'lon qilish
class DateClass // sinf a'zolari jimlik bo‘yicha yopiqdir
{
   int m_day; //jimlik bo‘yicha yopilgan, faqat sinfning boshqa a'zolari kirish huquqiga ega
   int m_month; // jimlik bo‘yicha yopilgan, faqat sinfning boshqa a'zolari kirish huquqiga ega
   int m_year; // jimlik bo‘yicha yopilgan, faqat sinfning boshqa a'zolari kirish huquqiga ega
};
int main()
```

```
{
   DateClass date;
   date.m_day = 12; // xato
   date.m_month = 11; // xato
   date.m_year = 2018; // xato
   return 0;
}
```

Siz ushbu dasturni kompilyatsiya qila olmaysiz, chunki sinfning barcha a'zolari jimlik bo‘yicha yopiqdir. Yopiq a'zolar (yoki "**private** a'zolar") - faqat shu sinfning boshqa a'zolari kira oladigan sinf a'zolari. **main() DateClass** a'zosi bo‘lmaganligi sababli, **date** obyekti xususiy a'zolariga kira olmaydi.

Sinf a'zolari jimlik bo‘yicha yopiq bo‘lsa-da, ularni public kalit so‘z yordamida ochiq qilishimiz mumkin:

```
class DateClass
{
   public: //
```

```
    int m_day; // ochiq, har qanday obyekt kirish huquqiga ega
    int m_month;
    int m_year;
};

int main()
{
    DateClass date;
    date.m_day = 3; //xato emas, chunki m_dayda public kirish
spetsifikatori mavjud
    date.m_month = 8;
    date.m_year = 2022;
    return 0;
}
```

DateClass a'zolari endi barcha obyektlar uchun ochiq bo‘lgani uchun ularga to‘g‘ridan -to‘g‘ri **main**() funksiyasidan kirish mumkin.

public kalit so‘z ikki nuqta bilan birgalikda *kirish spetsifikatori* deb ataladi. Kirish spetsifikatori ushbu spetsifikator a'zolariga qaysi obyekt kirishi

mumkinligini aniqlaydi. Har bir a'zo kirish spetsifikatoriga muvofiq kirish darajasini "oladi" (agar koʻrsatilmagan boʻlsa, standart kirish spetsifikatoriga koʻra).

C++ da 3 ta kirish darajasi mavjud:

1) *public* speksifikatori a'zolarni ochiq qiladi;
2) *private* speksifikatori a'zolarni yopiq qiladi;
3) *protected* spetsifikatori a'zolarga faqat doʻst va avlod sinflari uchun kirishni ochadi (bu haqida tegishli bobda batafsilroq ma'lumot beriladi).

Sinflarning ochiq a'zolari umumiy (yoki "public") interfeysni tashkil qiladi. Sinf a'zolariga sinfdan tashqaridan kirish mumkin boʻlganligi sababli, umumiy interfeys sinfdan foydalanadigan dasturlarning bir xil sinf bilan qanday aloqada boʻlishini aniqlaydi.

Ba'zi dasturchilar birinchi navbatda private-a'zolarni, keyin esa public-a'zolarni roʻyxatga olishni afzal koʻrishadi. Ular quyidagi mantiqni boshqaradilar: public a'zolar odatda private a'zolardan foydalanadilar (sinf metodlarida bir xil a'zo oʻzgaruvchilari), shuning uchun birinchi navbatda

private a'zolarni, keyin esa public a'zolarni aniqlash mantiqan to‘g‘ri keladi. Boshqa dasturchilarning fikricha, birinchi navbatda public a'zolari ko‘rsatilishi kerak. Bu yerda esa boshqa mantiq bor: private a'zolar yopiq va siz ularga to‘g‘ridan-to‘g‘ri kira olmaysiz, ularni birinchi o‘ringa qo‘yishingiz shart emas. Qaysi usuldan foydalanishni ixtiyoriy tanlash mumkin.

2-listingda berilgan dasturni ko‘rib chiqaylik:

```
#include <iostream>
class DateClass
{
 int m_day;
 int m_month;
 int m_year;
public:
 void setDate(int day, int month, int year)
 {
        m_day = day;
        m_month = month;
        m_year = year;
 }
 void print()
```

```
 {
        cout << m_day << "/" << m_month << "/" << m_year;
 }
 //Ushbu qo‘shimcha metodga e'tibor bering
 void copyFrom(const DateClass &b)
 {
        // b obyektining xususiy a'zolariga to‘g‘ridan -to‘g‘ri kirish mumkin
        m_day = b.m_day;
        m_month = b.m_month;
        m_year = b.m_year;
 }
};
int main()
{
  DateClass date;
  date.setDate(24, 12, 2022);
  DateClass copy;
  copy.copyFrom(date);
  copy.print();
  return 0;
}
```

C++ tilida koʻpincha e'tiborga olinmaydigan (notoʻgʻri tushuniladigan) bir nyuans shundaki, kirishni boshqarish obyekt emas, balki sinf asosida ishlaydi. Bu shuni anglatadiki, agar metod sinfning private a'zolariga kirsa, u ushbu sinfning har qanday obyektining private a'zolariga kira oladi.

Yuqoridagi misolda copyFrom() metodi DateClass sinfining a'zosi boʻlib, unga DateClass sinfining private a'zolariga kirish imkonini beradi. Bu shuni anglatadiki, copyFrom() faqat u bilan ishlaydigan yopiq obyektning private a'zolariga toʻgʻridan-toʻgʻri kira olmaydi (obyekt nusxasi), balki DateClass sinfining b obyekti xususiy a'zolariga ham toʻgʻridan-toʻgʻri kirishi mumkin.

Agar elementlarni sinfning bitta obyektidan bir xil sinfning boshqa obyektiga nusxalash zarur boʻlsa, bu foydali boʻladi. Bu haqda keyingi boblarda batafsil gaplashamiz.

KIRISH FUNKSIYALARI (GETTERLAR VA SETTERLAR)

Sinfga qarab, sinfning yopiq a'zolari o‘zgaruvchilarining qiymatlarini olish / o‘rnatish imkoniyatiga ega bo‘lish mumkin (sinf nima qilayotgani nuqtai nazaridan).

Kirish funksiyasi – bu qisqa umumiy funksiya, uning vazifasi sinfning yopiq a'zolarining o‘zgaruvchisini olish yoki o‘zgartirishdir. Masalan:

```
class MyString
{
private:
    char *m_string; // dinamik ravishda qatorni ajratish
    int m_length; // satr uzunligini aniqlash uchun o‘zgaruvchidan foydalanish
public:
    int getLength() { return m_length; } // m_length qiymatini olish uchun kirish funksiyasi
};
```

Bu yerda getLength () - bu m_length qiymatini qaytaradigan kirish funksiyasi.

Odatda kirish funksiyalari ikki xil bo‘ladi:

1) *getterlar* - bu sinfning yopiq a'zo o‘zgaruvchilarining qiymatlarini qaytaradigan funksiyalar;
2) *setterlar* - bu sinfning yopiq a'zo o‘zgaruvchilariga qiymatlarni belgilashga imkon beradigan funksiyalar.

3-listingda barcha xususiy a'zo o‘zgaruvchilari uchun getter va setterlardan foydalanadigan sinf keltirilgan.

```
class Date
{
private:
   int m_day;
   int m_month;
   int m_year;
```

```
public:
    int getDay() { return m_day; } // m_day uchun getter
    void setDay(int day) { m_day = day; } // m_day uchun setter

    int getMonth() { return m_month; } // month uchun getter
    void setMonth(int month) { m_month = month; } // month uchun
setter

    int getYear() { return m_year; } // year uchun gettter
    void setYear(int year) { m_year = year; } // year uchun setter
};
```

Bu sinfda hech qanday muammo yo‘q, shuning uchun foydalanuvchi to‘g‘ridan-to‘g‘ri ushbu sinfning xususiy a'zo o‘zgaruvchilarining qiymatlarini olishi yoki belgilashi mumkin, chunki to‘liq getterlar va setterlar to‘plami mavjud. MyString sinfidagi misolda, m_length o‘zgaruvchisi uchun setter ta'minlanmagan, chunki foydalanuvchi uzunlikni to‘g‘ridan-to‘g‘ri o‘rnatishga hojat yo‘q edi.

SINFLARDA INKAPSULYATSIYA PRINSIPINI QO‘LLASH

Nima uchun sinf a'zolarining o‘zgaruvchilarini yopiq qilish kerak?

Javob sifatida, o‘xshashlikdan foydalanaylik. Zamonaviy hayotda biz ko‘plab elektron qurilmalarga kirishimiz mumkin. Televizorda masofadan boshqarish pulti mavjud, uning yordamida siz televizorni yoqishingiz yoki o‘chirishingiz mumkin. Avtomobil haydash sizga ikki nuqta o‘rtasida tezroq harakatlanish imkonini beradi. Siz kamera yordamida suratga olishingiz mumkin.

Bu 3 narsaning hammasi umumiy shablondan foydalanadi: ular sizga ma'lum bir harakatni bajarish uchun oddiy interfeys (tugma, rulda va hokazo) beradi. Biroq, bu qurilmalarning aslida qanday ishlashi sizdan (foydalanuvchilar sifatida) yashirilgan. Masofadan boshqarish pultidagi tugmani bosish uchun siz televizor bilan ishlash uchun pultning "qopqog‘i ostida" nima bo‘layotganini bilishingiz shart emas. Avtomobilingizda gaz pedalini bosganingizda, yonish dvigateli g‘ildiraklarni qanday boshqarishini bilishning hojati yo‘q. Rasmga tushganda, sensorlar yorug‘likni pikselli tasvirga qanday yig‘ishini bilishning hojati yo‘q.

Interfeys va amalga oshirishni ajratish juda foydalidir, chunki bu obyektlarni ularning bajarilishini tushunmasdan ishlatishga imkon beradi. Bu ushbu qurilmalardan foydalanishning murakkabligini sezilarli darajada kamaytiradi va ularning sonini sezilarli darajada oshiradi (ular bilan o‘zaro aloqada bo‘lishingiz mumkin bo‘lgan qurilmalar).

Shunga o‘xshash sabablarga ko‘ra, dastur va interfeysni ajratish dasturlashda ham foydalidir.

Inkapsulyatsiya. Obyektga yo‘naltirilgan dasturlashda inkapsulyatsiya (yoki "axborotni yashirish") – bu obyektning bajarilish tafsilotlarini yashirish. Foydalanuvchilar obyektga umumiy interfeys orqali kirishadi.

C++ da inkapsulyatsiya kirish spetsifikatorlari orqali amalga oshiriladi. Odatda, sinfning barcha a'zolar o‘zgaruvchilari yopiqdir (amalga oshirish tafsilotlarini yashirish) va ko‘pchilik metodlar ochiq (foydalanuvchiga ochiq interfeys bilan). Foydalanuvchilardan umumiy interfeysdan foydalanishni talab qilish a'zo o‘zgaruvchilarini ochishdan ko‘ra qiyinroq bo‘lib tuyulishi mumkin, lekin u kodni qayta ishlatish va xizmat ko‘rsatishning yaxshilanishini ta'minlaydigan ko‘plab foydali afzalliklarni beradi.

Afzalliklar:

1) Inkapsulyatsiyalangan sinflarni ishlatish osonroq va dasturlarning murakkabligini kamaytiradi.

To‘liq qamrab olingan sinf bilan siz faqat qanday metodlar mavjudligini, ular qanday dalillar va qanday qiymatlarni qaytarishini bilishingiz kerak. Sinfning ichki qo‘llanilishini bilishning hojati yo‘q. Masalan, ismlar ro‘yxatini o‘z ichiga olgan sinf dinamik qator, C uslubidagi satrlar, array, vector, map, list yoki boshqa ma'lumotlar tuzilmasi yordamida amalga oshirilishi mumkin. Bu sinfdan foydalanish uchun uni amalga oshirish tafsilotlarini bilishingiz shart emas. Bu dasturlarning murakkabligini sezilarli darajada kamayadi va mumkin bo‘lgan xatolar soni ham kamayishi mumkin. Bu inkapsulyatsiyaning asosiy afzalligi.

C++ standart kutubxonasidagi barcha sinflar inkapsulyatsiyalangan. Tasavvur qiling, agar siz ularni ishlatish uchun string, vector yoki cout (va boshqa obyektlar) ning bajarilishini bilishingiz kerak bo‘lsa, C++ ni o‘rganish qanchalik qiyin bo‘lardi.

2) Inkapsulyatsiyalangan sinflar sizning ma'lumotlaringizni himoya qilishga va noto‘g‘ri ishlatilishining oldini olishga yordam beradi.

Global o‘zgaruvchilar xavflidir, chunki ularga kim kirishi va qanday ishlatilishiga qat'iy nazorat yo‘q. Ochiq a'zolar sinflari bir xil muammoga ega, faqat kichikroq miqyosda. Masalan, string sinfini yozishimiz kerak deylik. Biz quyidagilarni boshlashimiz mumkin:

```
class MyString
{
   char *m_string; // dinamik ravishda qatorni ajratish
   int m_length; // satr uzunligini kuzatish uchun o‘zgaruvchidan foydalanish
};
```

Ikki a'zo bog‘liq: m_length har doim m_string tutgan satr uzunligiga mos kelishi kerak. Agar m_length umumiy bo‘lsa, u holda har kim m_stringni o‘zgartirmasdan satr uzunligini o‘zgartirishi mumkin (yoki aksincha). Bu, albatta, muammolarga olib keladi. m_length va m_stringni yopiq qilib, foydalanuvchilar sinf bilan ishlash metodlaridan foydalanishga majbur bo‘ladilar.

Sinfni noto‘g‘ri ishlatishdan himoya qilish imkoniyati ham mavjud. Masalan, umumiy a'zo o‘zgaruvchiga ega bo‘lgan sinfni massiv sifatida ko‘rib chiqaylik (-listing).

```
class IntArray
{
public:
   int m_array[10];
};
```

Agar foydalanuvchilar massivga to‘g‘ridan-to‘g‘ri kira olsalar, ular noto‘g‘ri indeksdan foydalanishlari mumkin:

```
int main()
{
   IntArray array;
   array.m_array[16] = 2; // noto‘g‘ri indeks
}
```

Ammo, agar massivni xususiy qilsak, biz foydalanuvchini indeksning to‘g‘riligini tekshiradigan funksiyadan foydalanishga majburlashimiz mumkin:

```
class IntArray
{
private:
```

```
    int m_array[10]; // foydalanuvchi bu a'zoga to‘g‘ridan -to‘g‘ri kirish huquqiga ega emas

public:
    void setValue(int index, int value)
    {
        // Agar indeks noto‘g‘ri bo‘lsa, hech narsa qilmang
        if (index < 0 || index >= 10)
            return;

        m_array[index] = value;
    }
};
```

Shunday qilib, biz dasturimizning yaxlitligini himoya qilamiz.

3) Inkapsulyatsiyalangan sinflarni o‘zgartirish osonroq.

Quyidagi oddiy misol (-listing)ni ko‘rib chiqaylik:

```
#include <iostream>
class Values
{
public:
   int m_number1;
   int m_number2;
   int m_number3;
};
int main()
{
   Values value;
   value.m_number1 = 7;
   cout << value.m_number1 << '\n';
}
```

Bu dastur yaxshi ishlayotgan bo‘lsa-da, agar m_number1 nomini o‘zgartirishga yoki bu o‘zgaruvchining turini o‘zgartirishga qaror qilsak nima

bo‘ladi? Biz nafaqat bu dasturni, balki Values sinfidan foydalanadigan dasturlarning ko‘pini buzardik!

Inkapsulayatsiya, ularni ishlatadigan barcha dasturlarning ishlashini buzmasdan, sinflarning bajarilishini o‘zgartirish imkoniyatini beradi. - listingda m_number1 ga kirish uchun usullardan foydalanadigan, yuqoridagi sinfning inkapsulatsiyalangan versiyasi:

```
#include <iostream>

class Values
{
private:
   int m_number1;
   int m_number2;
   int m_number3;

public:
   void setNumber1(int number) { m_number1 = number; }
   int getNumber1() { return m_number1; }
```

```
};

int main()
{
   Values value;
   value.setNumber1(7);
   cout << value.getNumber1() << '\n';
}
```

Endi sinfning bajarilishini o‘zgartiraylik:

```
#include <iostream>
using namespace std;
class Values
{
private:
   int m_number[3]; // bu sinfning qo‘llanilishini o‘zgartiring

public:
```

```
    // Yangi dasturning ishlashi uchun metod o‘zgaruvchilarini yangilashimiz kerak
    void setNumber1(int number) { m_number[0] = number; }
    int getNumber1() { return m_number[0]; }
};

int main()
{
    // Lekin dasturimiz avvalgidek ishlashda davom etmoqda
    Values value;
    value.setNumber1(7);
    cout << value.getNumber1() << '\n';
}
```

E'tibor bering, sinfning umumiy interfeysida hech qanday funksiya prototipini o‘zgartirmaganimiz uchun, sinfdan foydalanadigan dasturimiz o‘zgarishsiz va muammosiz ishlashda davom etadi.

4) Inkapsulyatsiyalagan sinflarni tahrirlash qilish osonroq.

Nihoyat, inkapsulatsiya notoʻgʻri ketganda dasturlarni tuzatishga yordam beradi. Koʻpincha dasturning notoʻgʻri ishlashining sababi oʻzgaruvchilardan birining notoʻgʻri qiymati hisoblanadi. Agar har bir obyekt oʻzgaruvchiga toʻgʻridan-toʻgʻri kira oladigan boʻlsa, unda oʻzgaruvchining kodini oʻzgartirgan qismini kuzatish juda qiyin boʻlishi mumkin. Ammo, agar qiymatni oʻzgartirish uchun xuddi shu usulni chaqirishingiz kerak boʻlsa, ushbu usulda toʻxtash nuqtasini ishlatishingiz va noto'g'ri holatni koʻrmaguningizcha, har bir murojaat qiymatini oʻzgartirishi mumkin.