Ենթադասեր և ժառանգականություն

Դասի սահմանումը հնարավորություն է տալիս խուսափելու կրկնություններից: Այս եղանակով ծրագրային կոդը դառնում է ավելի հասկանալի և ընթեռնելի:

Օրինակ՝ կազմակերպության անդամներին հաշվառելու համար անհրաժեշտ է գրանցել յուրաքանչյուր աշխատակցի անձնական տվյալները: Որպեսզի գրանցումը ճիշտ կատարվի և ոչ մի տվյալ բաց չթողնվի, անհրաժեշտ է յուրաքանչյուր աշխատակցի համար ստեղծել քարտ, որտեղ էլ կգրանցվեն նրա տվյալները: Դրա համար անհրաժեշտ է սահմանել քարտի ձևաչափը՝ դասը:

Բոլոր աշխատակիցների համար պետք է գրանցվեն նույն անձնական տվյալները՝ անուն, ազգանուն, հայրանուն, ծննդյան տարեթիվ, հեռախոսահամար:

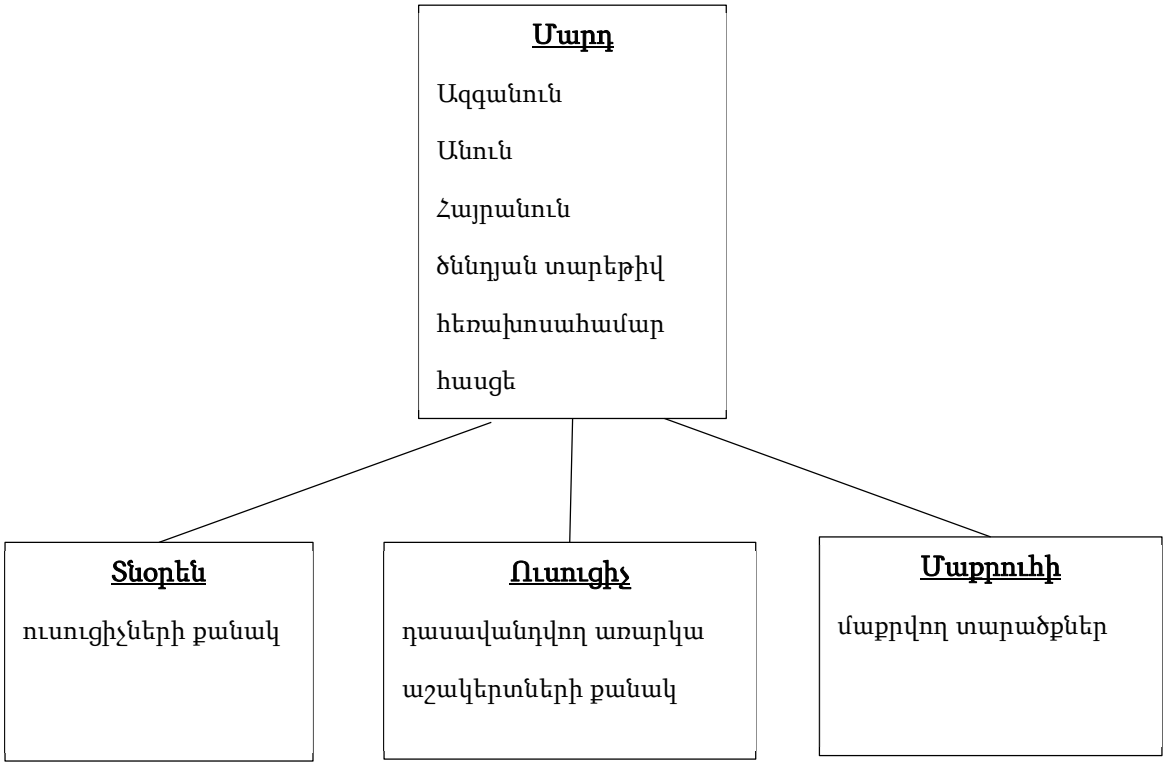
Բացի այս տվյալներից, կան նաև զուտ աշխատանքին առնչվող տվյալներ, որոնք տարբերվում են միմյանցից՝ կախված պաշտոնից:

Տնօրեն	**Ուսուցիչ**	**Մաքրուհի**
Ազգանուն	Ազգանուն	Ազգանուն
Անուն	Անուն	Անուն
Հայրանուն	Հայրանուն	Հայրանուն
ծննդյան տարեթիվ	ծննդյան տարեթիվ	ծննդյան տարեթիվ
հեռախոսահամար	հեռախոսահամար	հեռախոսահամար
հասցե	հասցե	հասցե
ուսուցիչների քանակ	դասավանդվող առարկա	մաքրվող տարածքներ
	աշակերտների քանակ	

Կարող ենք յուրաքանչյուր պաշտոնի համար սահմանել մի դաս, որը կպարունակի այդ դասի համար անհրաժեշտ տվյալները:

Բայց եթե ուշադիր նայենք այս քարտերին, ապա կտեսնենք, որ բոլորն էլ պարունակում են որոշ ընդհանուր տվյալներ, որոնք հատուկ են բոլոր մարդկանց՝ անկախ պաշտոնից: Սրանից հետևում է, որ կարելի է սահմանել մի ընդհանուր դաս՝ «մարդ», որը կպարունակի ընդհանուր տվյալները:

«Տնօրեն», «ուսուցիչ» և «մաքրուհի» դասերն արդեն կդառնան «մարդ» դասի ենթադասեր, որոնք կունենան «մարդ» դասի բոլոր հատկանիշները և էլի որոշ հատկանիշներ, որոնք հատուկ են միայն իրենց ենթադասին:



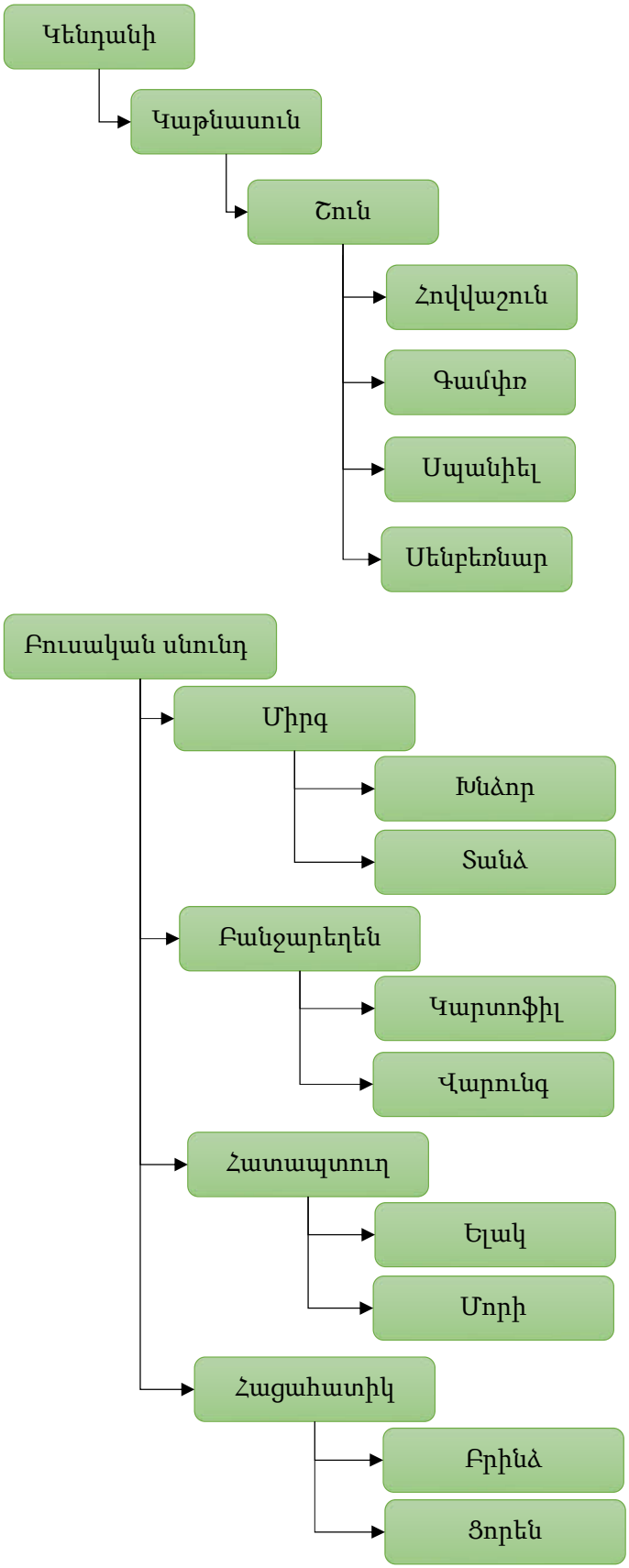
Ժառանգականությունը բոլոր հատկությունների և մեթոդների ժառանգումն է ծնող դասից:

Նոր դասը սահմանվում է արդեն գոյություն ունեցող (ծնող) դասի հիման վրա՝ ժառանգելով ծնող դասի բոլոր հատկանիշներն ու մեթոդները:

Ժառանգված դասից նույնպես կարող են ստեղծվել նոր դասեր:

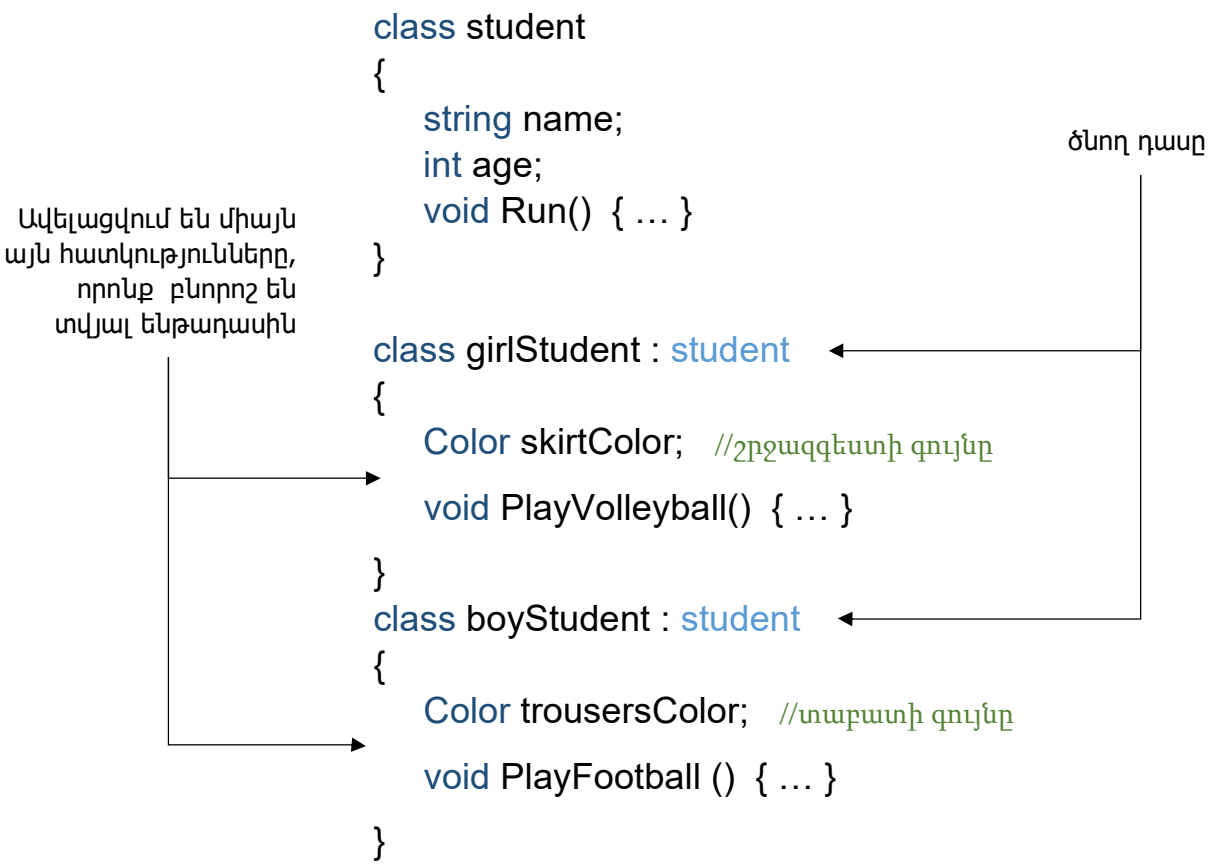
Օրինակ.

«Կենդանի» դասը ներկայացնում է բոլոր կենդանիների ընդհանուր նկարագիրը: Կենդանի դասի ենթադաս է «կաթնասուն» դասը, որը ժառանգում է «կենդանի» դասի բոլոր հատկությունները և ունի սեփական՝ միայն կաթնասուններին բնորոշ հատկություններ: «Շուն» դասը «կաթնասուն» դասի ենթադաս է, որը ժառանգում է «կաթնասուն» դասի բոլոր հատկությունները և ունի նաև միայն շանը բնորոշ հատկություններ: «Շուն» դասի ենթադասեր են «հովվաշուն», «գամփռ», «սպանիել», «սենբեռնար» դասերը:


Կենդանի
Կաթնասուն
Շուն
Հովվաշուն
Գամփռ
Սպանիել
Սենբեռնար
Բուսական սնունդ
Միրգ
Խնձոր
Տանձ
Բանջարեղեն
Կարտոֆիլ
Վարունգ
Հատապտուղ
Ելակ
Մորի
Հացահատիկ
Բրինձ
Ցորեն

Ենթադասերը C++ լեզվում

C++ լեզվում ենթադասը նկարագրելիս անվանումից հետո նշվում է ծնող դասը: Ենթադասը ժառանգում է ծնող դասի բոլոր հատկությունները (փոփոխականները) և վարքը (մեթոդները):



Այս օրինակում աշակերտի համար սահմանված են անունը, տարիքը և ֆիզկուլտուրայի դասին վազքի վարժանքը: Հանդիսանալով «աշակերտ» (student) դասի ժառանգ՝ և՛ աղջիկ աշակերտները (girlStudent), և՛ տղա աշակերտները (boyStudent) նույնպես ունեն անուն, տարիք և ֆիզկուլտուրայի դասին վազում են:

Բայց միայն աղջիկներն են, որ շրջազգեստով են և ֆիզկուլտուրայի դասի ընթացքում վոլեյբոլ են խաղում: Իսկ տաբատ հագնում են միայն տղաները և ֆիզկուլտուրայի դասի ժամանակ ֆուտբոլ են խաղում:

Ժառանգականություն

Գործնական աշխատանք

1. Նկարագրել «ուսուցիչ» և «աշակերտ» դասերը՝ ժառանգված «մարդ» դասից:
   - Ո՞ր տվյալներն են ժառանգվում «մարդ» դասից և որոնք են առկա միայն «ուսուցիչ» և «աշակերտ» դասերում:
   - Ո՞ր մեթոդներն են ժառանգվում «մարդ» դասից և որոնք են առկա միայն «ուսուցիչ» և «աշակերտ» դասերում:

2. Նկարագրել «ավտոմեքենա» դասը, որը որոշվում է ապրանքանիշով և շարժիչի հզորությամբ, ինչպես նաև ունի շարժիչի հզորությունը փոփոխելու հնարավորություն (մեթոդ):
   Նկարագրել «բեռնատար» դասը, որը «ավտոմեքենա» դասի ժառանգ է: Բեռնատարի համար ներկայացնել բեռի առավելագույն քաշը և այդ քաշը փոփոխելու հնարավորությունը (մեթոդ):

3. Նկարագրել «եռյակ» դասը: Այն ունի 3 բնական թվեր և այդ թվերը փոփոխող ֆունկցիաներ (մեթոդներ):
   Նկարագրել «եռանկյունի» դասը, որը «եռյակ» դասի ժառանգ է: Լրացնել «եռանկյունի» դասի մեթոդները՝ եռանկյան պարագիծը և մակերեսը հաշվող ֆունկցիաներով:

4. Նկարագրել «զույգ» դասը: Այն ունի 2 բնական թվեր և այդ թվերը փոփոխող ֆունկցիաներ (մեթոդներ):
   Նկարագրել «ուղղանկյուն» դասը, որը «զույգ» դասի ժառանգ է: Լրացնել «ուղղանկյուն» դասի մեթոդները՝ ուղղանկյան պարագիծը և մակերեսը հաշվող ֆունկցիաներով:

5. Նկարագրել «զույգ» դասը: Այն ունի 2 ամբողջ թվեր և այդ թվերը փոփոխող ֆունկցիաներ (մեթոդներ):
   Նկարագրել «փող» դասը, որը «զույգ» դասի ժառանգ է: Ամբողջ թվերից մեկը ցույց է տալիս դրամի, իսկ մյուսը՝ լումայի քանակը: Ավելացնել «փող» դասում տրված իրական թվով (ամբողջ մասը՝ դրամ, կոտորակային մասը՝ լումա) գումարը ավելացնելու և պակասեցնելու ֆունկցիաները:

Ինկապսուլյացիա

ՕԿԾ-ում ամեն օբյեկտ ունի իր հատկություններն ու վարքը:

Օրինակ՝ «ինքնաթիռ» օբյեկտը ունի իրեն բնորոշող հատկություններ՝ գույն, ծավալ, տարողունակություն, հզորություն և վարք, կարող է թռչել, կարող է վայէջք կատարել:

Այսպիսի մոտեցումն օգնում է կառուցելու բարդ համակարգեր: Տրոհելով մոդելավորվող միջավայրը օբյեկտների և սահմանելով դրանց միջև կապեր և փոխհարաբերություններ՝ կարողանում ենք ավելի լավ պատկերացնել մոդելավորվող միջավայրը:

Այս մոտեցման կարևոր սկզբունքներից է տվյալ օբյեկտին վերաբերող հատկություններն ու գործողությունները օբյեկտի նկարագրության մեջ ամփոփելը:

Ինկապսուլյացիան օբյեկտի վարքի ամփոփումն է օբյեկտի ներսում:

Օրինակ՝ «օդաչու» օբյեկտը չպետք է իմանա, թե ինչպես է թռչում «ինքնաթիռ» օբյեկտը: Նրանց փոխհարաբերությունները կառուցվում են՝ միմյանց ազդակներ ուղարկելով: «Օդաչու» օբյեկտը, սեղմելով անհրաժեշտ կոճակները, ազդակ է ուղարկում «ինքնաթիռ» օբյեկտին այն մասին, որ նա պետք է թռչի: Թռչելու համար նախատեսված գործողությունները «ինքնաթիռ» օբյեկտը կատարում է իր ներսում: «Օդաչու» օբյեկտն այդ գործողությունների հետ չի առնչվում:

Ի՞նչ է տալիս ինկապսուլյացիան

1. Ներքին տվյալների պաշտպանություն
   Եթե օբյեկտի որևէ տվյալ կամ մեթոդ ստեղծված է զուտ օբյեկտի ներքին աշխատանքի համար, ապա պետք չէ, որ այլ օբյեկտները տեսնեն այդ տվյալներն ու մեթոդները:
2. Անվտանգություն
   Եթե տվյալները պաշտպանված են, ապա կարող եք վստահ լինել, որ այլ օբյեկտ չի կարող փոփոխել դրանք, ինչը կարող է հանգեցնել աշխատանքի խափանման կամ սխալների:

   Տվյալների պաշտպանության համար սահմանվում է տվյալների հասանելիության երեք մակարդակ.
   - ✓ public (բաց)
     Այս կերպ նկարագրված տվյալներն ու մեթոդները հասանելի են բոլորի համար: Այլ օբյեկտները կարող են և՛ տեսնել, և՛ փոփոխել տվյալները, և՛ օգտագործել մեթոդները:
   - ✓ private (փակ)
     Այս կերպ նկարագրված տվյալներն ու մեթոդները հասանելի են միայն այդ օբյեկտի ներսում: Դրանք չեն կարող տեսնել այլ օբյեկտները:
   - ✓ protected (պաշտպանված)
     Այս կերպ նկարագրված տվյալներն ու մեթոդները հասանելի են միայն այդ օբյեկտի և նրա ժառանգների համար:

```
// «ուղղանկյուն» դասի սահմանումը
class Rectangle
{
    private int left;     // ուղղանկյան դիրքը ձախից
    private int top;    // ուղղանկյան դիրքը վերևից
    protected int width;  // ուղղանկյան լայնությունը
    protected int height; // ուղղանկյան բարձրությունը
    public Color col;     // ուղղանկյան գույնը
}
```

Կոդ 1

Կոդ 1 օրինակում նկարագրված է «ուղղանկյուն» դասը, որի համար սահմանված են դիրքը ցույց տվող փոփոխականներ (left, top), չափը ցույց տվող փոփոխականներ (width, height) և գույնը ցույց տվող փոփոխականը (col):

Ստեղծենք այս դասի օբյեկտ:

```
Void Main
{
    Rectangle rect;
    rect.left = 10;    // հնարավոր չէ (rect օբյեկտին չպատկանող ֆունկցիայում չի
                                թույլատրվում փոփոխել արժեքը)
    rect.top = 10;  // հնարավոր չէ
    rect.width = 25;   // հնարավոր չէ (արժեքը կարելի է փոփոխել միայն rect օբյեկտի
                                ներսից կամ նրանից ժառանգված օբյեկտից)
    rect.height = 40;  // հնարավոր չէ
    rect.col = red;   // հնարավոր է  (այս փոփոխականը հասանելի է ամեն տեղից)
}
```

Կոդ 2

Ընդլայնենք Rectangle դասի նկարագրությունը և ավելացնենք մի ֆունկցիա, որը տալիս է ուղղանկյան դիրքը:

```
// «ուղղանկյուն» դասի սահմանումը
class Rectangle
{
    private int left;     // ուղղանկյան դիրքը ձախից
    private int top;      // ուղղանկյան դիրքը վերևից
    protected int width;  // ուղղանկյան լայնությունը
    protected int height; // ուղղանկյան բարձրությունը
    public Color col;     // ուղղանկյան գույնը

    public SetPosition (int x, int y)
    {
        if (x>=0 && y>=0)
            {
                left = x;
                top = y;
            }
    }
}
```

Կոդ 3

Main ֆունկցիայում փոփոխենք ուղղանկյան դիրքը՝ օգտագործելով SetPosition մեթոդը:

```
Void Main
{
    Rectangle rect;
    rect.col = red;   // հնարավոր է  (այս փոփոխականը հասանելի է ամեն տեղից)
    rect.SetPosition(10, 10);
}
```

Կոդ 2

Այս եղանակով հնարավոր է փոփոխել ուղղանկյան դիրքը, քանի որ SetPosition ֆունկցիայի հասանելիությունը նշված է public: Իսկ SetPosition ֆունկցիան գործում է տվյալ օբյեկտի ներսում և կարող է փոփոխել բոլոր տվյալները:

Մեթոդի միջոցով տվյալների փոփոխության կազմակերպումը հնարավորություն է տալիս խուսափելու սխալներից:

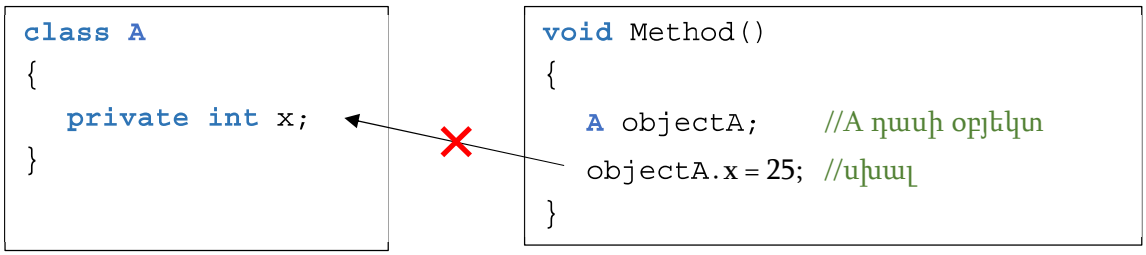
Օրինակ՝ եթե left և top փոփոխականները լինեին հասանելի բոլորի համար, ապա այդ դեպքում հնարավոր չէր լինի վերահսկել, որ այդ փոփոխականներին բացասական արժեք չվերագրվի: SetPosition ֆունկցիան ամեն վերագրումից առաջ ստուգում է վերագրվող արժեքների համապատասխանությունը և միայն ճիշտ արժեքների դեպքում է կատարում վերագրումը:

Private հասանելիություն

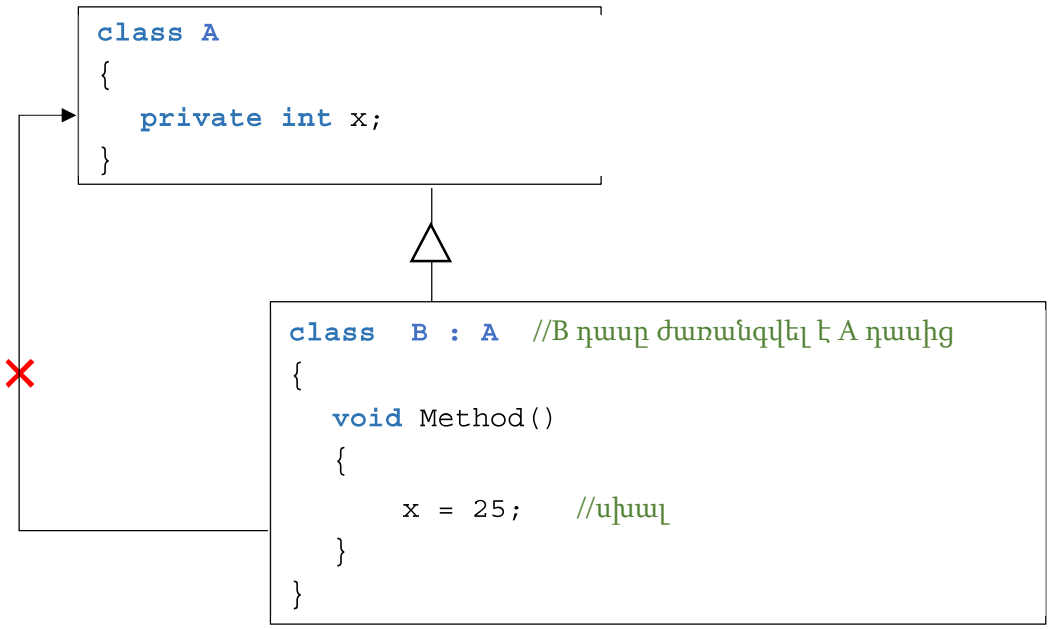
Կարելի է տեսնել և փոփոխել այդ դասի ներսում:

```
class A
{
   private int x;

   void Method()  //մեթոդը նկարագրված է A դասի ներսում
   {
      x = 25;  //թույլատրվում է

      A objectA;       //A դասի օբյեկտ
      objectA.x = 40;  //թույլատրվում է
   }
}
```

Չի կարելի տեսնել կամ փոփոխել այդ դասին չպատկանող ֆունկցիայում:



Չի կարելի տեսնել կամ փոփոխել այդ դասից ժառանգված դասում:



Public հասանելիություն

Կարելի է տեսնել կամ փոփոխել տվյալ դասի ներսում:

```
class A
{
   public int x;

   void Method()  //մեթոդը նկարագրված է A դասի ներսում
   {
      x = 25;  //թույլատրվում է

      A objectA;      //A դասի օբյեկտ
      objectA.x = 40;  //թույլատրվում է
   }
}
```

Կարելի է տեսնել կամ փոփոխել այդ դասին չպատկանող ֆունկցիայում:

```
class A
{
   private int x;
}
```

```
void Method()
{
   A objectA;      //A դասի օբյեկտ
   objectA.x = 25;  //թույլատրվում է
}
```

Կարելի է տեսնել կամ փոփոխել այդ դասից ժառանգված դասում:

```
class A
{
   private int x;
}
```

```
class   B : A   //B դասը ժառանգվել է A դասից
{
   void Method()
   {
        x = 25;     //թույլատրվում է
   }
}
```

Protected հասանելիություն

Կարելի է տեսնել կամ փոփոխել տվյալ դասի ներսում:

```
class A
{
   public int x;

   void Method()  //մեթոդը նկարագրված է A դասի ներսում
   {
      x = 25;  //թույլատրվում է

      A objectA;       //A դասի օբյեկտ
      objectA.x = 40;  //թույլատրվում է
   }
```

Չի կարելի տեսնել կամ փոփոխել այդ դասին չպատկանող ֆունկցիայում:

```
class A
{
   private int x;
}
```

```
void Method()
{
   A objectA;       //A դասի օբյեկտ
   objectA.x = 25;  //թույլատրվում է
}
```

Կարելի է տեսնել կամ փոփոխել այդ դասից ժառանգված դասում:

```
class A
{
   private int x;
}
```

```
class   B : A   //B դասը ժառանգվել է A դասից
{
   void Method()
   {
        x = 25;     //թույլատրվում է
   }
}
```

Ինկապսուլյացիա

Գործնական աշխատանք

6. Նկարագրել A դասն այնպես, որ main ֆունկցիան լինի անսխալ:
   Ի՞նչ կգրվի էկրանին main ֆունցիայի աշխատանքից հետո:

```
void main ()
{
    A obj1;
    obj1.x = 3;
    obj1.y = 5;
    cout << obj1.x * obj1.y << endl;
}
```

7. Նկարագրել A դասն այնպես, որ main ֆունկցիան լինի անսխալ:
   Ի՞նչ կգրվի էկրանին main ֆունցիայի աշխատանքից հետո:

```
void main ()
{
    A obj1;
    obj1.x = 3;
    obj1.y = 5;
    cout << obj1.x * obj1.y << endl;
}
```

8. Արդյոք անսխա՞լ կաշխատի main ֆունկցիան: Եթե ոչ, ապա ինչո՞ւ:

```
Class I
{
    private:
        int i=0;

    public:
        int Get()
```

```
        {
            return i;
        }
};

void main()
{
    I i1;
    I i2;
    i2.i += 400;
}
```

9. Ի՞նչ ցույց կտա էկրանին main ֆունկցիան:

```
Class I
{
    private:
        int i=20;

    public:
        int Get()
        {
            return i;
        }
};

void main()
{
    I i1;
    I i2;
    cout << i1.Get() + i2.Get() << endl;
 }
```

Պոլիմորֆիզմ

Պոլիմորֆիզմը ՕԿԾ-ի երեք հիմնական սկզբունքներից մեկն է. ժառանգականություն, ինկապսուլյացիա, պոլիմորֆիզմ: Պոլիմորֆիզիմը նմանատիպ ֆունկցիաները նույն անունով ներկայացնելն է:

Դիտարկենք այն դեպքը, երբ կայքում հնարավոր է կատարել 3 տիպի հրապարակում՝ նորություն, հայտարարություն և հոդված: Այս 3 հրապարակումներն էլ ունեն նույն կառուցվածքը՝ վերնագիր, բովանդակություն: Բայց կան նաև տարբերություններ. հոդվածն ունի հեղինակ, նորությունը՝ աղբյուր, իսկ հայտարարությունը՝ վերջնաժամկետ:

Ելնելով այս տարբերություններից՝ կարելի է նկարագրել 3 տարբեր դասեր.

- Հոդված
    - վերնագիր
    - բովանդակություն
    - հրապարակել հոդվածը (հեղինակ)
- Նորություն
    - վերնագիր
    - բովանդակություն
    - հրապարակել նորությունը (աղբյուր)
- Հայտարարություն
    - վերնագիր
    - բովանդակություն
    - հրապարակել հայտարարությունը (վերջնաժամկետ)

Սա անհարամարություն է ստեղծում ծրագրավորողի համար, որը պետք է կարողանա կողմորոշվել դասերի բազմազանության մեջ:

Այս մոտեցմամբ, բոլոր այն օբյեկտները, որոնք գրեթե նույնն են իրենց իմաստով ու էությամբ, բայց ունեն չնչին տարբերություն, պետք է նկարագրվեն տարբեր դասերով: Դա բարդացնում ու խճճում է աշխատանքը:

Խառնաշփոթից խուսափելու համար ՕԿԾ-ում կիրառվում է պոլիմորֆիզմի սկզբունքը:

Պոլիմորֆիզմ անվանումն առաջացել է հունարեն Πολύμορφος բառից, ինչը թարգմանաբար նշանակում է բազում ձևեր:

Նմանատիպ օբյեկտների համար նկարագրվում է մի դաս:

- Հրապարակում
    - վերնագիր
    - բովանդակություն
    - հրապարակել ()

Դասի նկարագրության մեջ «հրապարակել» ֆունկցիայի սահմանումը տրվում է աբստրակտ՝ չկոնկրետացված:

Հետագայում ենթադասերը սահմանելիս կոնկրետացվում է «հրապարակել» ֆունկցիայի սահմանումը:

- Հոդված: Հրապարակում
  հրապարակել (հեղինակ)
- Նորություն: Հրապարակում
  հրապարակել (աղբյուր)
- Հայտարարություն: Հրապարակում
  հրապարակել (վերջնաժամկետ)

Պոլիմորֆիզմը նույն անվան օգտագործումն է նմանատիպ ֆունկցիաները նկարագրելու համար: Այդ ֆունկցիաները միմյանցից տարբերվում են միայն պարամետրերով:

Պոլիմորֆիզմը C++ լեզվում

Ծնող դասի նկարագիրը

```
class A
{
    public:
        virtual void f() =0;  //սահմանվում է վիրտուալ f ֆունկցիան
}
```

Կոդ 1

Ժառանգ դասերի նկարագիրը

```
class B: A    //B դասը հանդիսանում է A ժառանգ
{
  public:
    void f() //նկարագրվում է f ֆունկցիան B դասի համար
    {
        cout << "B" << endl;
    };
}

class C: A    //C դասը հանդիսանում է A ժառանգ
{
  public:
    void f() {};   //նկարագրվում է f ֆունկցիան C դասի համար
            {
        cout << "C" << endl;
          };
}
```

Կոդ 2

Վիրտուալ ֆունկցիայի օգտագործումը

```
void main()
{
    A* b = new B();
    A* c = new C();
    b->f();   //այստեղ կանչվում է B դասի ֆունկցիան
    c->f();   //այստեղ կանչվում է C դասի ֆունկցիան
}
```

Կոդ 3