*Т.А. Гуржій,* асп.
(Національний технічний університет
"Київський політехнічний інститут"),
*М.С. Львов,* канд. фіз.-мат. наук, доц.,
*О.В. Співаковський,* канд. фіз.-мат. наук, доц.
(Херсонський державний педагогічний університет)

# ФУНКЦІОНАЛЬНА КЛАСИФІКАЦІЯ ОБ'ЄКТІВ ПРОГРАМНИХ СИСТЕМ

**Анотація.**

У роботі запропоновано класифікацію об'єктів програмних систем, яка спирається на їх функціональне призначення в системі,

розглянуті основні проблеми, що виникають при реалізації таких об'єктів та запропоновані шляхи їх розв'язання. Наведена нами класифікація є уточненням класифікації [3].

## 1. Вступ

### 1.1. Об'єкти та їх описи

Об'єктно-орієнтоване програмування — це методологія програмування, що заснована на представленні програми у вигляді сукупності *об'єктів*, кожний з яких є реалізацією певного *класу* (типу особливого виду).

Об'єктами є, наприклад, точки декартової площини, кулі для гри в більярд, вікна, використовувані для введення-виведення інформації, різного роду пристрої, призначені для обробки інформації (калькулятори, перекладачі, броузери тощо), розклад занять факультету вищого навчального закладу, список викладачів кафедри та ін.

Об'єкт описується своїми *атрибутами (інформаційними полями)* і *методами*.

### 1.2. Класи

Класи (об'єктні типи) призначені для опису сімейств об'єктів, що в системі грають однакові ролі. Тому всі об'єкти того самого класу характеризуються однаковими (однотипними) наборами атрибутів (інформаційних полів). Термін "атрибути" підкреслює змістовну сторону цього поняття, а термін "інформаційні поля" – синтаксичну. Однак, об'єднання об'єктів у класи визначається не тільки наборами атрибутів, але і поводженням. Так, наприклад, об'єкти "крапка площини" і "вектор" можуть мати однакові атрибути – координати. При цьому вони можуть відноситися до одного класу, якщо розглядаються в задачі просто як пари чисел, або до різних класів, якщо для рішення задач системи вектори потрібно складати, а крапки переміщати.

Відзначимо, що необхідно розрізняти термін "клас" (об'єктний тип) і термін "екземпляр класу" (об'єкт) точно так само, як розрізняються поняття "тип" даних і "перемінна" даного типу.

### 1.3. Імена об'єктів, атрибутів і методів

Імена програмних об'єктів у методології об'єктно-орієнтованого програмування відіграють особливу роль. Ім'я структурного компонента програми, крім синтаксичного і прагматичного

навантаження, повинне нести ще і максимальне семантичне навантаження. Це означає, що воно повинно бути не тільки синтаксично правильним і унікальним у своїй області видимості, але і максимально точно позначати той фізичний об'єкт, інформаційна модель якого описується. Варто пам'ятати, що створюється програмний компонент, яким повинні користуватися інші учасники робочих груп проекту.

### 1.4. Класифікація атрибутів об'єкта

Інформаційні поля призначені для збереження інформації про об'єкт та використовуються як самим об'єктом, так і іншими об'єктами. З погляду використання атрибути об'єкта можна класифікувати в такий спосіб:

• *атрибути – ідентифікуючі характеристики* об'єкта. Ідентифікуючі атрибути об'єкта – це ті дані, що характеризують власне об'єкт. Розрізнення цих атрибутів означає відмінність об'єктів. Ідентифікуючі атрибути об'єкта, отже, приписуються йому в момент створення (народження). Вони не повинні мінятися протягом життєвого циклу цього об'єкта. Сукупність таких атрибутів називають ідентифікатором об'єкта. Ідентифікатор об'єкта повинен бути унікальним. Якщо серед атрибутів об'єкта можна вибрати декілька груп атрибутів, що ідентифікують об'єкт, одна з таких груп повинна бути призначена ідентифікатором "по означенню". У цьому випадку говорять про привілейований ідентифікатор об'єкта;

• *атрибути – описові параметри* об'єкта. Описові параметри об'єкта описують його стан у даний момент часу. Вони можуть змінюватися протягом його життя.

Так, в описі об'єкта TBilliardBall (більярдна куля) незмінними залишаються такі атрибути, як радіус, номер, колір і маса. Це – ідентифікуючі властивості. Атрибути, що змінюються – координати центра, вектор швидкості. Це – описові атрибути (параметри) більярдної кулі. Очевидно, що ідентифікатором більярдної кулі як об'єкта є його номер. Незважаючи на те, що радіус і маса не міняються протягом життя кулі, вони не ідентифікують його як об'єкт, хоча б тому, що їхні чисельні значення, швидше за все , однакові для всіх більярдних куль. Деякі кулі можуть бути розфарбовані однаково, тому колір кулі тільки розділяє (факторизує) групу куль на декілька одноколірних груп.

З іншого боку, центр кулі може ідентифікувати його як об'єкт, оскільки дві кулі не можуть знаходитися в одній крапці більярдного

столу. Однак, оскільки координати кулі змінюються, цей атрибут не є ідентифікатором "по означенню".

**Приклад 1.** Об'єктний тип Більярдна куля.

```
Type
  BilliardBall = object
        Number : TballNumber;   {ідентифікатор}
        Center  : TPoint;       {параметр-опис}
        Radius  : Real;         {ідентифікуюча характеристика}
        Color   : Tcolor;       {ідентифікуюча характеристика}
        Massa   : Real;         {ідентифікуюча характеристика}
        Velosity : Tvector      {параметр-опис}
    {опис методів}
end;
```

• *допоміжні атрибути* (інформаційні поля – посилання). Інформаційні поля посилального типу реалізують атрибути, спеціально призначені для установки і підтримки зв'язків між об'єктами. Ці зв'язки називають залежностями, а відповідні атрибути – допоміжними. (Термін "залежність" запозичений з теорії баз даних. З математичної точки зору залежність – це відношення на множині об'єктів).

Наприклад, відношення "людина X – батько людини Y" визначено на множині людей і встановлює залежність батьківства між деякими парами людей. Відношення "місто X – столиця країни Y" установлює залежність між елементами різних безлічей – безлічі міст і безлічі країн.

**Приклад 2.** Допоміжні атрибути. Залежності.

```
Type
    TPerson = object
        Name : Tname;
        Father : ^TPerson;      {допоміжний атрибут}
        {опис атрибутів}
        {опис методів}
    end;

    TSity = object
        Name : Tname;
        {опис атрибутів}
        {опис методів}
    end;
```

```
TLand = object
    Name : Tname;
    Capital : ^TSity;          {допоміжний атрибут}
    {опис атрибутів}
    {опис методів}
end.
```

У типізованих мовах програмування кожне дане, описане і використане в програмі, має свій тип. Тому інформаційні поля, що відповідають атрибутам класифікуються, перед усім, своїми типами. З цього погляду, інформаційні поля можуть бути:

- логічними;
- перелічувальними;
- інтервальними;
- числовими;
- символьними;
- строковими;
- структурами;
- посиланнями;
- об'єктами.

Перелічуються типи, що використовуються програмістом для опису об'єктних типів, використовуються, перед усім, для опису атрибутів – якісних характеристик об'єктів. Інтервальні типи використовують для контролю діапазонів відповідних даних (визначення доменів).

```
Type
  Sex = (Man, Women);
  WeekDay = (Monday, Wednesday, . . .);
  Month = 1..12;
```

Логічні поля визначають логічні властивості об'єктів, числові – їхні чисельні параметри і характеристики.

Строкові поля несуть текстову інформацію про об'єкт. Часто зустрічаються ситуації, коли значення строкових полів можна так чи інакше закодувати. Числовий код відповідного атрибута тоді включається в список атрибутів, а операції кодування і декодування реалізуються методами цього об'єкта.

*Області визначення об'єктів називають доменами, отже, визначають безлічі припустимих значень атрибута.*

Визначення домену насамперед більш загальне поняття, ніж його тип, що оголошується в описі. Домени можуть визначатися ще і

так званим *інваріантом* (наприклад, посиланням на так чи інакше реалізоване перерахування своїх значень або правилом, що визначає допустимість атрибута і реалізованим у виді логічної функції).

Наприклад, множина значень атрибута Місто_країни може зберігатися в спеціальному документі Список_міст.

Правилом, що визначає домен атрибута Центр об'єкта Більярдна куля, є система нерівностей

0<= X.Center <= BilliardTableSize, 0 <= Y.Center <= BilliardTableSize.

Відзначимо, що інваріанти можуть установлювати домени не тільки окремих атрибутів, але і груп атрибутів і всього об'єкта. Так, якщо більярдний стіл має форму кола, інваріантом є правило: "Центр кулі належить колу радіуса BilliardTable.Radius."

Комбіновані й інші складені типи в описах об'єктів відіграють особливу роль. Відзначимо, перед усім, що записи, масиви "у чистому виді" в описи об'єктів звичайно не включають. Справа в тім, що кожне дане, якщо воно включено в опис об'єкта, повинне оброблятися, відповідно до дисципліни об'єктно-орієнтованого програмування, тільки сукупністю специфічних методів. Це означає, що атрибут-запис можна і потрібно замінити атрибутом-об'єктом. Включення в структуру об'єкта іншого об'єкта називають *агрегацією або композицією*.

Сказане вище в значній мірі відноситься й іншим структурним типам мови. Якщо дане структуроване, завжди існує необхідність визначення операцій його обробки.

### 1.5. Методи

Як уже відзначалося, найважливішим нововведенням, що характеризує ОО методологію програмування, є включення у визначення об'єкта операцій, що може виконувати цей об'єкт.

Під методами в ООП розуміють алгоритми, спрямовані на виконання об'єктом операцій (дій) як по запиту іншого об'єкта, так і для "задоволення своїх власних потреб" в обробці інформації.

Таким чином, методи об'єктів в ООП служать тим же цілям, що і процедури (функції) в структурному програмуванні. Однак, на відміну від структурного програмування, в якому для рішення підзадачі основна програма (чи процедура) викликає відповідну процедуру, передаючи їй необхідні дані і приймаючи результати обробки у виді параметрів, ОО підхід припускає, що кожен об'єкт, реагуючи на повідомлення-запит від іншого об'єкта, вирішує поставлену задачу,

виконуючи необхідну операцію власним методом. Таким чином, для будь-якої дії (операції), виконуваного над атрибутами об'єкта, повинен бути написаний окремий метод.

Процедурний стиль програмування пропонує створювати і використовувати процедури і функції для опису алгоритмів обробки даних. ОО стиль програмування пропонує створювати і використовувати методи обробки даних, збережених (інкапсульованих) як атрибути об'єкта. Використання методів об'єкта принципово відрізняється від викликів чи процедур функцій.

Семантична відмінність полягає в тім, що в тілі опису методу спрощений доступ до інформаційних полів відповідного об'єкта.

### 1.6. Модулі й опис об'єктів

Концепція програмного модуля системи і реалізація модулів (Units) у цьому середовищі відповідає принципу інкапсуляції об'єктно-орієнтованого програмування. Справді, опис об'єктних типів передують їхньому використанню в інших програмних модулях проектованої системи. У цих модулях вони описуються як окремі сутності – перемінні відповідного об'єктного типу, оброблювані методами цього типу.

Оскільки використання засобів модуля неможливо здійснити "за замовчуванням" і програмісту модулів верхнього рівня доступні тільки ті методи обробки об'єктних змінних, котрі в модулі використовуваного об'єкта описані як інтерфейсні, саме модуль є тією капсулою, що значною мірою захищає об'єкти від некоректних дій програміста. Нарешті, об'єктний клас, реалізований у виді модуля, може використовуватися і повторно – при проектуванні інших програмних систем.

### 1.7. Інкапсуляція

Г.Буч [ 2] визначає реальний об'єкт як сутність, що має чітко визначені границі. Об'єкт володіє станом, поведінкою й ідентичністю; структура і поводження об'єктів визначають загальний для них клас. Це визначення має яскраво виражений філософський характер, оскільки воно оперує філософськими категоріями "поведінка", "ідентичність" та ін.

Вище ми "визначили" програмні об'єкти як інформаційні моделі реальних об'єктів і процесів. Звичайно, це теж ще не означення в математично строгому розумінні цього терміна, а тільки пояснення, що розкриває тільки одну прагматичну сторону цього поняття. Саме, як

інформаційна модель, об'єкт "зсередини" описується такими даними й операціями їхньої обробки, що за поводженням об'єкта дозволяють ідентифікувати його з абстракцією реального об'єкта.

Означення об'єкта стає математично точним тільки в рамках формальної системи, такий, як конкретна об'єктно-орієнтована система програмування.

Однак, незалежно від характеру означень, усі вони описують об'єкт як єдність даних і операцій, відокремлюючи при цьому зовнішні прояви цієї єдності у вигляді поведінки від внутрішнього пристрою – структури, що визначає поведінку. Таким чином, існують чіткі границі між пристроєм об'єкта і його поводженням. Поведінка об'єкта описує його з абстрактної точки зору, а структура об'єкта визначає механізм цього поводження. Поділ описів поведінки і структури називають інкапсуляцією.

*Інкапсуляція – це процес відділення один від одного елементів опису об'єкта, що визначають його пристрій і поведінку; інкапсуляція служить для того, щоб ізолювати контрактні зобов'язання абстракції від їхньої реалізації.*

Практично це означає, що опис об'єкта повинен бути розділений на дві частини – інтерфейс, що визначає поведінку об'єкта, і реалізацію, що визначає механізми досягнення бажаного поводження.

### 1.8. Загальнодоступні та приватні атрибути і методи

Описи об'єктних типів, у свою чергу, можуть бути розділені на так звані приватні, захищені і загальнодоступні частини.

Таким чином, опис об'єктного типу може мати три розділи: розділ відкритого сегмента об'єкта, що містить описи загальнодоступних атрибутів і методів, розділ захищеного сегмента і розділ приватного сегмента – з описами приватних атрибутів і методів.

Методологія ОО проектування орієнтована на розробку великої і функціонально складної комп'ютерної системи. Такі програми проектуються колективом розроблювачів. Тому методологія ОО проектування зобов'язана підтримувати поділ праці.

Припустимо, що розроблювальна програмна система складається з декількох компонентів, кожну з яких пише свій програміст. Що він повинен знати про інші компоненти? Тільки їхню правильну поведінку, тобто повідомлення-запити на виконання завдань, якими можна користуватися і реакції іншого компонента на ці повідомлення. Протокол програмного компонента-об'єкта, отже, єдине необхідне її зовнішній опис. Все інше повинно бути приховано і надійно захищено.

Т.Бадд [1] приводить наступні правила, названі принципами Парнасу:

- розроблювач програмного компонента повинний представляти користувачу цього компонента всю інформацію, що потрібна для ефективного її використання, і нічого крім цього;
- розроблювач програмного забезпечення, що використовує програмний компонент-додаток, повинен знати тільки необхідне поводження компонента, і нічого крім цього.

Відзначимо, що цей важливий принцип програмування непросто усвідомити програмістам, що працюють поодинці над розробкою невеликої за розміром програми. Найбільшу складність у таких програмах можуть представляти проблеми реалізації віртуозних алгоритмів, що вирішують конкретні задачі обробки даних. Насправді основною проблемою тут є проблема стикування програмних компонентів, розроблювальних незалежно. Рішенню цієї проблеми присвячені всі принципи об'єктно-орієнтованого програмування, у тому числі і принципи виділення інтерфейсної і реалізаційної частин, сформульовані тут.

Програміст компонента може експериментувати над її реалізацією, поліпшуючи працюючі усередині алгоритму й удосконалюючи внутрішні структури даних, не торкаючись при цьому інтерфейсного протоколу. Зовнішній світ, тобто інші програмні компоненти при цьому не зачіпаються, тому поводження всієї системи в цілому не міняється.

## 2. Фунціонально спеціалізовані об'єкти

Будь-яка складна програмна система являє собою сукупність взаємодіючих об'єктів тієї чи іншої структури і складності. Методи визначення більш складних об'єктів з більш простих, методи взаємодії об'єктів ми розглянемо пізніше. Очевидно, що будь-які такі побудови містять у собі опис конкретних об'єктів, які мають просту будівлю і спеціалізовану поведінку. Об'єкти, про які піде мова, описують на заключному етапі проекту – етапі, на якому приймаються чисто технічні, остаточні рішення. Деякі приклади таких об'єктів ми зараз розглянемо. Наша мета полягає в тому, щоб одержати більш глибокі представлення про ті конкретні проблеми, що виникають і зважуються на цьому етапі. Приведена нижче класифікація не є ні стандартною, ні повною, але в достатній мірі відображає мету.

### 2.1. Атрибути об'єктів і об'єкти-атрибути

Об'єкти-атрибути призначені для збереження і спеціалізованої обробки даних. Як правило, кожен такий об'єкт поєднує в собі дані, що описують одну з комплексних характеристик конкретного реального об'єкта. З іншого боку, ця характеристика може використовуватися для визначення багатьох об'єктних типів. Тому вона є самостійною об'єктною одиницею і повинна бути описана окремо. Наприклад, об'єкт TDate (дата) використаний в описі об'єкта TPerson. Але, зовсім очевидно, що цей об'єктний тип може і повинен бути включений у визначення всіх класів, для опису яких вимагаються дати. До об'єктів-атрибутів належать, наприклад, такі об'єкти, як TPerson, TPostAdress, TVector тощо. Імена об'єктів-атрибутів входять у словник проекту програмної системи як іменника-ідіоми.

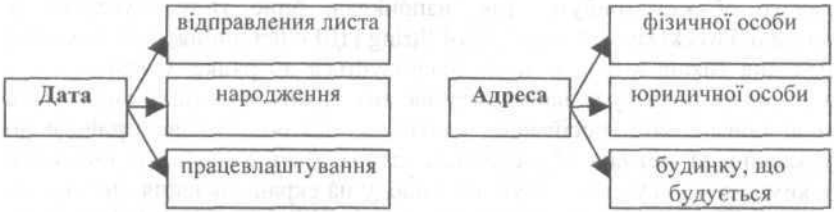


**Рис 1.** Абстракції і спеціалізації об'єктів

Властивість універсальності (багаторазової використованості) об'єктів-атрибутів, мабуть, є найбільш важливим із практичної точки зору. Його необхідно забезпечувати в першу чергу. Це досягається, насамперед, відповідним рівнем абстракції опису об'єкта.

Атрибути об'єкта повинні описувати тільки ті його характеристики, що формують відповідну даному типу абстракцію, тобто є найбільш загальними і специфічними в цій абстракції.

Так, атрибутами класу TPerson повинні бути такі характеристики, як ім'я (привілейований ідентифікатор), підлога, можливо, ще деякі інші атрибути. Найдальшому використанні тип TPerson доповнюється іншими характеристиками, що визначають ролі людини.


**Рис 2.** Абстракції і рольові спеціалізації об'єктів

Другим важливим моментом в описі атрибутів є дотримання відповідного рівня абстракції в їхній типізації.

Типи атрибутів повинні забезпечувати відповідний рівень абстракції і максимальні зручності для специфічних операцій обробки даного об'єкта-атрибута. Так, наприклад, опис типу TweekDay у вигляді TweekDay = Array[1..7] of String [10] є неприйнятним, тому що ім'я дня тижня в цьому описі прив'язується до рядка. Очевидно, що значенням поля буде найменування дня тижня на якійсь конкретній мові (англійській, російській, суахілі,..), що обов'язково приведе до ускладнення методів обробки його даних. Тип TWeekDay, описаний таким чином, зручний тільки для виводу на екран значення дня тижня, а операція виводу на екран не є специфічною для цього класу. Нормальним у цьому описі є використання перечислювального типу TDayName.

TweekDay = Array[1..7] of TDayName

### 2.2. Методи доступу

*Характерними операціями об'єктів – атрибутів є операції доступу до інформаційних полів цього об'єкта.*

Оскільки при реалізації об'єктних типів дотримується принцип приховання даних, власне інформаційні поля об'єкта недоступні користувачу об'єкта. Тому для кожного атрибута, яким можна користуватися, повинні бути визначені операції доступу для читання і запису. Методи цих операцій називають відповідно методами-селекторами і методами-модифікаторами. Найбільш простим селектором є операція читання, а модифікатором – операція запису. Звичайно селекцію (доступ-читання) реалізують методами з іменами типу Get<Ім'я_атрибута>. Наприклад, для класу TDate повинні бути визначені методи GetDay, GetMonth, GetYear, GetWeekDay. Методи-селектори найкраще реалізовувати як функції:

```
Function TDate.GetDay : Tday;
begin
    GetDay := Day;
end.
```

Коли тип інформаційного поля не дозволяє визначити метод-селектор як функцію, використовують процедури:

```
Procedure TMan.GetName(var F : Tname);
begin
  F := Name
 end.
```

Модифікації атрибутів об'єкта частіше реалізують за допомогою операції з іменами-дієсловами типу Put<Ім'я атрибута>, Set<Ім'я атрибута>. Наприклад:

```
Procedure TDate.PutDay (DD: Tday);
begin
    Day := DD;
end.
```

Оскільки нове значення атрибута імпортується, методи-модифікатори реалізують як процедури з відповідним параметром.

Правила захисту інформаційних полів об'єкта від некоректних змін жадають від програміста чіткого поділу всіх атрибутів об'єкта на дві групи:

- атрибути "тільки для читання" – визначені тільки методи-селектори (типу Get);
- атрибути "для запису і читання" – визначені методи-селектори і методи-модифікатори (типу Put…,Set…)...

### 2.3. Властивості й атрибути

Принципи приховання й інкапсуляції означають, що будь-який об'єкт може бути ідентифікований "у зовнішньому світі", тобто використаний іншими об'єктами тільки відповідно до протоколу, в якому перераховані і специфіковані його операції (у вигляді заголовків методів). Операції – селектори виділяють у протоколі об'єкта його властивості.

*Властивістю об'єкта називають дане, що представляє інформацію про даний об'єкт і доступне іншим об'єктам.*

Отже, існує принципове розходження між атрибутами і властивостями об'єкта. Властивості є зовнішнім проявом набору атрибутів об'єкта і його зв'язків із зовнішнім світом.

Як приклад розглянемо дві можливі реалізації класу TDate.

1. Об'єктний тип TDate реалізований за допомогою системної функції, що повертає системну дату у форматі <MM-DD-YY-WD>, де MM – номер місяця, DD – день місяця, YY – рік, WD – назва дня тижня англійською мовою. У цій реалізації тип TDate атрибутів не має.

2. Об'єктний тип TDate реалізований набором атрибутів Month – номер місяця, Day – день місяця, Year – рік.

Нехай і в першій, і в другій реалізації визначені ті самі операції з іменами методів GetDay, GetWeekDay, GetMonth, GetYear. З погляду зовнішнього користувача ці об'єктні типи тотожні. Вони володіють тими самими властивостями. Це дійсно дві різні реалізації того самого типу, тому що вони представлені одним протоколом. Оскільки в конкретній програмній системі об'єктний тип ідентифікується ім'ям, ці реалізації альтернативні. Програміст може реалізувати тільки один варіант типу TDate.

Помітимо, що в першому варіанті операції-модифікатори можуть бути реалізовані тільки за допомогою іншої системної функції – функції зміни системної дати. Це – дуже небезпечна операція, що повинна бути заборонена для виконання в прикладних програмах. Якщо ж у програмній системі дійсно необхідні і поточна дата, і деякі інші дати, необхідно визначити два різних класи TCurrentDate і TSomeDate, один з яких обробляє системну дату методами-селекторами, а інший додатково визначений ще і методами-модифікаторами, що змінюють атрибути TDate. Однак, крім операцій доступу, у додатках можуть знадобитися й інші операції над датами, реалізація яких не залежить від того, з яким типом дати вони виконуються. Такий об'єктний тип необхідно створювати окремо.

*Сукупність методів об'єктного типу містить методи особливого типу, що визначають властивості об'єктів. Властивості об'єктів ідентифікують об'єкти для зовнішніх користувачів.*

### 2.4. Основні і похідні атрибути

Використовуючи математичну термінологію, властивість Property найпростішого об'єкта-атрибута Obj можна визначити як функцію від його атрибутів $Atr_1$, $Atr_2$, . . ., $Atr_k$ як від змінних:

$Obj.Property = F(Obj.Atr_1, Obj.Atr_2, ..., Obj.Atr_k)$

Тому сукупність $Atr_1$, $Atr_2$, . . . , $Atr_k$ атрибутів об'єкта повинна задовольняти вимоги *функціональної повноти*.

*Функціональна повнота набору атрибутів означає, що будь-яка властивість об'єкта може бути виражена у виді функції від його атрибутів, що може бути обчислена.*

Наприклад, атрибутами вектора на площині можуть бути його Декартові координати, оскільки відомо, що будь-які інші характеристики і властивості векторів виражаються через їхні Декартові координати.

Вимога функціональної повноти набору атрибутів об'єкта грає важливу методологічну роль. Їм потрібно керуватися при визначенні набору атрибутів. Тоді при реалізації методів не виникне ніяких проблем принципового характеру.

З властивістю функціональної повноти тісно зв'язана властивість не залежності набору атрибутів.

*Набір атрибутів називають функціонально незалежним, якщо жоден з атрибутів цього набору не може бути обчислений через інші атрибути.*

Наприклад, набір атрибутів Day, Month, Year, WeekDay є функціонально залежним, оскільки день тижня WeekDay можна обчислити, знаючи значення Day, Month, Year.

Атрибути функціонально незалежного набору називають *основними*, а інші – *похідними*.

Як правило, у наборі всіляких даних, які можна розглядати як кандидатів в атрибути об'єкта, виділяють тільки функціонально незалежний набір. Дані цього набору визначають як атрибути, а інші дані реалізують як властивості, описуючи методи доступу до них у виді обчислень. Однак, іноді, має сенс доповнити функціонально незалежний набір атрибутів декількома похідними атрибутами, якщо це сильно вплине на якість програми (наприклад, дозволить різко збільшити її швидкодію). У цьому випадку реалізація методів-селекторів ускладнюється, оскільки в кожному з них потрібно реалізувати обчислення значень похідних атрибутів. Розглянемо приклад.

**Приклад 3.**

Припустимо, що в деякій програмі потрібно визначити такий об'єкт TPerson, для якого одними з основних операцій були б операції визначення батька і матері цього об'єкта.

```
TPerson = object
```

```
  private
    FamilyName,
    OwnName,
    FatherName : Tname;
    Father,
    Mother : ^TPerson;
  public
    Procedure GetFamilyName(var P : Tname);
    Procedure GetOwnName(var P : Tname);
    Procedure GetFatherName(var P : Tname);
    Procedure GetFuther(var P : TPerson);
    Procedure GetMother(var P : TPerson);
end.
```

У цьому описі сукупність атрибутів є функціонально залежною, оскільки значення атрибута FatherName може бути обчислене через заслання-значення поля Father^.OwnName. Отже, інформаційне поле FatherName можна виключити. Однак, при цьому, програмісту доведеться реалізовувати далеко не простий алгоритм побудови по батькові на прізвище, в основі якого лежать спеціальні правила російської мови. Виникає законне питання, чи варто це робити, якщо можна просто включити до списку атрибутів ще один похідний атрибут, поклавши рішення мовних проблем на користувача програми.

У більш загальній формі, властивості об'єктів залежать ще й від властивостей інших об'єктів як від параметрів. Наприклад, об'єктний тип TPerson ми описали, використовуючи зв'язок з об'єктами того ж типу. Тим самим методи визначення властивостей TPerson можуть використовувати і дані об'єктів Father, Mother.

### 2.5 Об'єкти-обчислювачі

Основною функцією ще одного типу найпростіших об'єктів - об'єкта-обчислювача є реалізація одного чи декількох обчислювальних алгоритмів. Об'єкти-обчислювачі іноді називають віртуальними машинами.

У найбільш простій ситуації об'єкт-обчислювач являє собою реалізацію одного, але, як правило, досить складного обчислення. Це може бути, наприклад, ефективний алгоритм сортування масивів (швидке сортування), що у системі використовують декілька об'єктних типів. Інший приклад – генератор – об'єкт-генератор випадкових чисел із заданою функцією розподілу. Опис об'єктів такого типу містить

відповідний обчислювальний метод. Об'єкти-обчислювачі як атрибути можуть мати такі поля, як *Точність_обчислень*, *Кількість_ітерацій*, і т.д. Наприклад:

```
Type
    TSorter = object
        QuickSort(var A:array of Integer; First, Last: Index);
    end.
```

Тут First і Last – відповідно перший і останній індекси діапазону , що сортується в масиві A.

```
    TRandomGenerator = object
            Genetate( N : Integer );
    end.
```

Тут N – кількість випадкових чисел у вибірці.

У більш загальному випадку об'єкти-обчислювачі можуть містити в собі методи реалізації декількох обчислювальних алгоритмів з однієї предметної області. Розглянемо як приклад такої предметної області аналітичної геометрії на площині, що оперує з точками, відрізками, прямими. Характерним для цієї предметної області є те, що її операції визначені над різнотипними об'єктами. Тому має сенс визначити декілька об'єктних типів-атрибутів, інкапсулюючі в них тільки операції ініціалізації, доступу, а потім визначити об'єкт – предметну область, описавши там так звані багатосортні операції, що вирішують основні задачі відповідного розділу аналітичної геометрії.

**Приклад 4.** Аналітична геометрія.

```
Type
  TPoint = object
      private
      X, Y : Real;
      public
      function Get : Real;
      function Get : Real;
      { інші методи }
  end.
```

```
TSegment = object
    private
      A, B : TPoint;
    public
     function Get : Real;
     function Get : Real;
     { інші методи }
end.

TLine = object
    private
      CoefA, CoefB, CoefC : Real;
    public
      function GetCoefA : Real;
      function GetCoefB : Real;
      function GetCoefC : Real;
      { інші методи }
end.
TGeometry = object

                          {Розподіл відрізка в даному відношенні}
    Procedure DivideSegment(s : TSegment; lambda : Real; var
A:Tpoint);
                          {Довжина відрізка}
    Function GetSegmentLendth(s : Tsegment) : Real;
                          {Формування відрізка AB}
    Procedure MakeSegment(A, B : TPoint; var l : TSegment);
                          {Рівняння прямої, що проходить через A, У}
    Procedure MakeLine(A, B : TPoint; var l : TLine);
                          {Крапка перетинання двох прямих}
    Procedure IntersectLines(l, p : TLine; var A : TPoint);
                          {Кут між двома прямими}
    Function GetAngle(l, p : Tline) : Real;
          {інші методи рішення задач аналітичної геометрії }
end.
```

### 2.6. Зовнішні об'єкти

Істотну роль у проектуванні комп'ютерної системи грає її інтерфейс користувача.

Під інтерфейсом користувача програмної системи розуміють сукупність способів взаємодії користувача з програмною системою і засобів програмної системи, за допомогою яких ці способи реалізовані.

Як і інші модулі програмної системи, інтерфейсна її частина проектується у виді системи об'єктів, взаємодіючих як один з одним, так і з об'єктами внутрішньої частини (ядра) програмної системи. Ці об'єкти називають зовнішніми чи інтерфейсними.

Призначення зовнішнього об'єкта – реалізація операцій виводу-введення-висновку, пошуку й інших операцій взаємодії користувача з програмною системою.

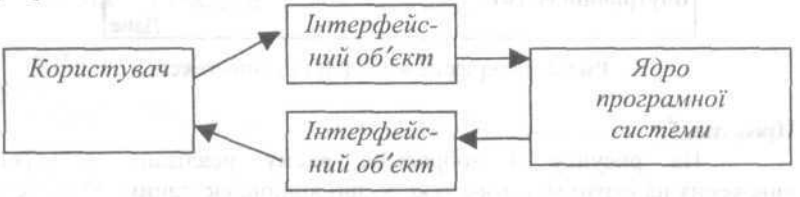


**Рис 3.** Схема інтерфейсу користувача

Проектування навіть найпростіших зовнішніх об'єктів має ряд особливостей. Основна з них полягає в тому, що програміст повинен визначити особливого типу об'єкт "Користувач" і описати протокол взаємодії цього об'єкта з проектованим інтерфейсним об'єктом. Оскільки реальний користувач працює на комп'ютері, застосовуючи пристрій виводу-введення-висновку, операції користувача є деяким обмеженням набору операцій відповідного пристрою. Кожен пристрій має свій ресурс, частина якого віддається об'єкту. Таким чином, при проектуванні виводу-введення-висновку програміст вирішує задачу розподілу ресурсів зовнішніх пристроїв і керування ними.

Ще одна особливість проектування інтерфейсних пристроїв полягає в тім, що типи (формати) даних, підтримуваних зовнішніми пристроями, відрізняються в загальному випадку від форматів даних – атрибутів "внутрішнього об'єкта", з якими взаємодіє інтерфейсний об'єкт. Тому повинні бути реалізовані операції перетворення типів даних з одного представлення в інше.

Таким чином, інтерфейсний об'єкт, як будь-який пристрій сполучення, має двосторонню, "двошарову" структуру – одна сторона "обслуговує" користувача, а інша – внутрішній об'єкт.


**Рис 4.** Інтерфейс виведення Дане-текст

**Приклад 5.**

На рисунку 4 зображено схему реалізації інтерфейсу виведення на екран монітора тексту, що відповідає даним, збереженим внутрішнім об'єктом.

Розглянемо більш детально ті проблеми, що виникають при реалізації цієї схеми.

**Розподіл ресурсів зовнішнього пристрою.**

На екрані монітора Об'єкт Поле виведення тексту виглядає так:

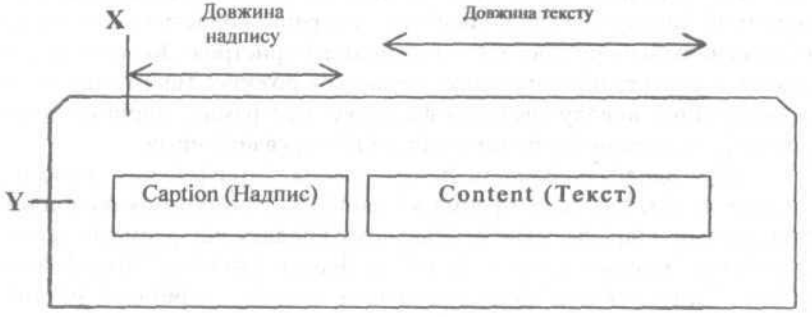


**Рис 5.** Розташування об'єкта Поле виведення на екрані

Об'єкт Поле виведення тексту представлений наступним описом:

**Клас** : Поле виведення тексту **Тип** : TTextField

| **Атрибути** | | | |
|---|---|---|---|
| **Імена** | **Типи** | **Зміст** | |
| Xpos | TsizeX | Координата X позиції поля на екрані | |
| Ypos | TsizeY | Координата Y позиції поля на екрані | |
| Caption | TStringN | Напис до тексту | |
| CaptionLen | TTextSize | Довжина напису до тексту | |
| CaptionColor: | Byte; | Колір напису до тексту | |
| TextLen | TTextSize | Довжина тексту, виведеного на екран | |
| TxtColor : | Byte | Колір тексту | |
| FieldColor : | Byte | Колір тла текстового поля | |
| **Методи** | | | |
| **Імена** | **Параметри** | **Типи** | **Зміст** |
| Init | Всі атрибути | | Ініціює об'єкт |
| Done | | | Видаляє об'єкт з екрана |
| Print | Content | TString | Виводить текст на екран |
| Clear | | | Очищає поле тексту |

Метричні атрибути об'єкта розподіляють екран, обмежуючи розміри об'єкта, інші атрибути визначають колірну гаму об'єкта. Атрибут Caption ідентифікує даний екземпляр об'єктного типу. Помітимо, що якщо розглядати опис цього об'єкта атрибутами, у ньому можна виділити три найпростіших об'єкти-атрибути: об'єкти Позиція на екрані, Напис і Текст – Зміст.

**Перетворення форматів даних**

У тих випадках, коли внутрішній об'єкт у системі унікальний, операції перетворювача можна реалізувати методами цього об'єкта. Нехай, наприклад, у системі передбачене спеціальне вікно для видачі різних повідомлень про події, що відбуваються в процесі роботи системи. Кожна з цих подій має свій "внутрішній" код. Проблему

дешифрування коду – перетворення його в текстове повідомлення – повинен вирішувати той об'єкт, в якому ця подія відбулася.

В іншому варіанті, коли унікальним є зовнішній об'єкт, операції перетворювача, навпаки, приписують цьому зовнішньому об'єкту. Наприклад, поточну дату на сторінці барвисто оформленого календаря потрібно переводити методами власне календаря, оскільки об'єкт Поточна Дата унікальна і тільки для цих цілей і створений.

Нарешті, у тих випадках, коли власне перетворення є універсальною операцією, що обслуговує декілька різнотипних зовнішніх і внутрішніх об'єктів, доцільно визначити об'єкт-обчислювач, що здійснює ці перетворення.

Наприклад, у всіх фінансових документах загальний підсумок необхідно друкувати особливим способом – ціле число основних грошових одиниць – прописом, а "копійок (центів), ..." – у виді двозначного числа. У цьому випадку доцільно визначити окремий об'єкт-обчислювач, оскільки і документів, і форм роздруківки підсумкових сум у них може бути кілька.

**Приклад 6. Редактор рядка**

Як приклад об'єкта, що здійснює введення даних, розглянемо об'єкт Редактор рядка. Операції цього об'єкта повинні підтримувати процес редагування рядка тексту. На рисунку 6 представлено схему реалізації інтерфейсу редактора рядка.

Розподіл ресурсу клавіатури означає приписування кожної з клавіш деякої операції редагування. Обмеження набору операцій полягає в блокуванні деяких груп клавіш.


**Рис 6.** Схема реалізації інтерфейсу редактора рядка

**Клас : Редактор рядка** **Тип : TstringEditor**

| **Інформаційні поля** | | | |
|---|---|---|---|
| **Імена** | **Типи** | | **Зміст** |
| EditString | String | | Текст, що редагується |
| Cursor | Byte | | Позиція курсора |
| Mode | (Insert, OwerWrite) | | Режим вставки/заміни |
| **Методи** | | | |
| **Імена** | **Параметри** | **Типи** | **Зміст** |
| Init | Str | String | Початок редагування Str |
| GoLeft | | | Курсор уліво |
| GoRight | | | Курсор управо |
| DeleteRight | | | Видалення символу праворуч |
| DeleteLeft | | | Видалення символу ліворуч |
| ChangeMode | | | Переключення режиму Ins/Qwr |
| AddChar | Ch | Char | Вставка/напис символу |
| Done | | | Кінець редагування |
| Monitor | | | Розпізнавання натиснутої клавіші |

Метод Monitor виконує особливу роль: він розпізнає, яка клавіша натиснута і, у залежності від цього, доручає виконання своєї операції тому чи іншому методу.

### 2.7. Керуючі об'єкти і методи

*Керуючими об'єктами (методами), називають об'єкти, методи, що здійснюють керування системою чи її компонентами.* Для позначення керуючих об'єктів використовуються також терміни контролер, монітор та ін. Типовим прикладом реального об'єкта-монітора є панель керування будь-якою багатофункціональною інтерактивною системою. (Комп'ютер, телевізор, автомобіль, залізниця, й інші системи, що оснащені такими панелями, хоча вони можуть по-різному називатися). При проектуванні таких систем центральну роль відіграють об'єкти-монітори, що реалізують функції

керування. Помітимо, що на відміну від інших об'єктів, програмні об'єкти-монітори і системи керування часто вже не моделюють реальні керуючі об'єкти, а самі виступають у цій ролі. У цьому і полягає комп'ютеризація керування.

Функція керування, як відомо, визначає керуючий вплив на об'єкт керування ( чи систему її компонент) у залежності від впливів зовнішнього середовища, об'єкта керування і свого власного стану. Правильний розподіл обов'язків у тріаді "зовнішнє середовище – монітор – об'єкт керування" – основна задача, розв'язуване програмістом на етапі аналізу проектованої системи.

На рисунку 7 показано одну зі стандартних схем керування – схема керування з зворотним зв'язком.


**Рис 7.** Керування з використанням зворотного зв'язку

**Керуючі методи**

Відзначимо, що в керуванні мають потребу не тільки складні системи чи її компоненти, але часто і самі об'єкти. Характерним прикладом такого об'єкта є розглянутий вище редактор рядка. Керування ним здійснює метод Monitor зовнішнім середовищем тут є клавіатура, її сигнали – натискання клавіш. Об'єкт керування – рядок, що редагується. Керуючі впливи – зміни в рядку, що редагується. Результат натискання клавіші може бути різним у залежності від параметрів рядка, що редагується, (наприклад, її довжини) і стану самого монітора. У нашому прикладі стан монітора визначається атрибутом (Insert – Owerwrite). У більш просунутих редакторах стан визначають ще і такі атрибути, як режим "прописні-рядкові букви", мова, розкладка тощо. Оскільки в редакторі рядка й об'єкт керування, і монітор реалізовані в одному об'єкті StringEdit, розподіл обов'язків

"Керування – об'єкт керування" здійснюється на рівні атрибутів і методів. Метод Monitor керує редагуванням, атрибути типу (Insert – Owerwrite) визначають безліч керуючих станів об'єкта. Атрибут EditString відіграє роль об'єкта керування, а керуючі впливи реалізовані схованими методами типу GoLeft.

**Керуючі об'єкти**

*Важливим методологічним аспектом при проектуванні і реалізації керування є той факт, що операції керування мають більший ступінь спільності, чим це необхідно для керування конкретною системою чи об'єктом.*

Наприклад, основні операції керування рухом графічного об'єкта в ігровому полі конкретної комп'ютерної гри власне кажучи не залежать чи залежать у малому ступені від зовнішнього вигляду об'єкта. Справді, переміщення об'єкта – це зміна координат однієї (початкової) точки ПРО, поворот – перетворення координат іншої точки (поворот вектора ОА). Тому можна визначити абстракції "траєкторія руху", "обертання", "зміна розмірів" тощо. Другий приклад – операції керування редагуванням рядка й операції керування однорівневим меню, які можна об'єднати в абстракції "редагування послідовності".

Тому один з основних методів проектування керування – *використання загальних принципів керування у виді абстрактних об'єктів і їх модифікація стосовно до конкретних об'єктів і систем керування.*

Вибір реалізації керування обумовлений багатьма факторами, але основні з них – ступінь складності керованої системи чи компоненти, а також – її ступінь залежності по керуванню від зовнішнього середовища.

Один з можливих підходів до реалізації керування складається в описі, так званого, керуючого автомата.

Як ми вже відзначали, керуючий вплив залежить від зовнішнього середовища, об'єкта керування і стану монітора. Надалі ми не будемо виділяти об'єкт керування з множини об'єктів, що належать до зовнішнього середовища. Справді, цей об'єкт, як і інші об'єкти, є зовнішнім стосовно монітора, а в задачах керування системою в цілому він просто збігається з цим середовищем.

Таким чином, можна визначити дві функції, значення яких визначають функціонування монітора: функцію виходів $\gamma$ і функцію переходів $\delta$.

$$\gamma: <S, X> \dashrightarrow Y$$
$$\delta: <S, X> \dashrightarrow S$$

Тут:

**X** – множина впливів зовнішнього середовища (вхідних сигналів);

**Y** – множина керуючих впливів (вихідних сигналів);

**S** – множина керуючих станів монітора;

Елементи множини **X** називають *подіями*, а елементи множини **Y** – *діями*. Керування реалізують за допомогою спеціального атрибута, значення якого називають *керуючими станами* монітора.

У множині **S** виділений елемент $s_0$, названий початковим станом, а також множина $S^*$, названа множиною заключних станів.

Таким чином, що керує автомат визначається набором

$$\Re = < X, Y, S, \delta, \gamma, s_0, S^* >$$

множина **X** подій (сигналів зовнішнього середовища) описується кінцевим набором імен властивостей, методи яких реалізують запити до зовнішніх об'єктів. У нашому прикладі монітора редактора рядка це системна функція ReadKey.

Множина дій **Y** описується кінцевим набором імен операцій керування, методи яких реалізують повідомлення об'єкта чи керування декільком об'єктам, що утворять керований компонент чи систему. У нашому прикладі монітора редактора рядка це – набір схованих методів типу GoLeft.

Нарешті, множина **S** керуючих станів призначена для "запам'ятовування" історії функціонування монітора. Саме, виконавши дію **y**, керуючий автомат переходить у новий стан **s**, запам'ятовуючи тим самим факт виконання дії. Як ми вже з'ясували, що керують станами в нашому прикладі є значення атрибута Mode.

Таким чином, проектування монітора полягає в описі поводження керуючого автомата $\Re$. При цьому подіями (елементами множини **X**) є властивості об'єктів зовнішнього середовища, а дії – елементи **Y** – суть найменування методів ( можливо, з параметрами). Множина **S** керуючих станів визначається кінцевим набором атрибутів монітора.

*Реалізація керування полягає у визначенні множини **S** і функції переходів* $\delta$ *на цій множині.*


**Рис.8.** Автомат керування процедурою покупки товару

Розглянемо приклад реалізації керування процедурою покупки товару. У цій торговій операції беруть участь покупець, магазин і банк. Керування операцією децентралізовано: кожна сторона контролює правильність виконання своїх дій. З погляду покупця операція представляється послідовністю станів, представлених на рисунку 8.

Вхідні сигнали – елементи множини **X** підрозділяються на дві групи: команди користувача і повідомлення магазина і банка.

Команди користувача: продовжити операцію; скасувати операцію.

Повідомлення магазина: прайс-лист представлено; товар відсутній; рахунок представлений; гроші отримані; документи на одержання товару отримані.

Повідомлення банку: рахунок клієнта заблоковано; гроші перераховані; грошей на рахунку недостатньо.

Одна з теорем теорії автоматів затверджує, що для будь-якого автомата, визначеного функціями переходів і виходів $\delta(s, x)$, $\lambda(s, x)$ можна визначити еквівалентний автомат, функція виходів якого має вид $\lambda(s, x) = \mu(\delta(s, x))$.

Це означає, що можна так визначити множину керуючих станів монітора **S** і функцію переходів **δ**(s, x), що вибір методу з множини **Y** буде цілком визначатися станом **s** монітора. Отже, у загальному виді керування можна реалізувати як вибір нового керуючого стану, перехід у який супроводжується керуючим впливом:

```
Case δ(s, x) of
   s0  : Object1.y1;
   s1  : Object2.y2;
    .....
   s*  : Objectk.yk;
end.
```

Розглянемо деякі варіанти цієї загальної схеми:

1. Монітори з одним керуючим станом – найбільше простій і разом з тим часто застосовуваний варіант керування.

```
Case x of
  X1 : Object1.y1;
  .....
  Xm : Object1.ym;
end.
```

Метод ExternalObject.GetVariant, звертаючись до навколишнього середовища, генерує номер вихідного сигналу – елемента множини виходів

**Y** = <Object1.Method1, ..., Objectk.Methodk>.

У випадку, коли після звертання до одного з методів **Y** керування знову повинне здійснювати даний монітор, оператор вибору замикається в нескінченний цикл, причому принаймні один з методів повинен реалізувати вихід з цього циклу.

```
Repeat
 Case ExternalObject.GetVariant(Parameter) of
   Variant1 : Object1.Method1(Parameter);
  .....
   Variantk : Objectk.Methodk(Parameter);
 end;
until False;
```

2. Монітори з вектором керуючих станів – ще один простий варіант керування, що часто зустрічається. У цьому варіанті керування описується вектором, кожна координата якого реалізує значення одного з параметрів керування. У розглянутому прикладі редактора

рядка вектор **S** керування просунутої версії редактора має три координати:

**S** = <Insert, Caps Lock, Language>

У принципі можливі наступні варіанти реалізації таких моніторів:

- хешування;
- вкладене розгалуження;
- комбінування хешування і вкладеного розгалуження.

При цьому керування може здійснювати як сам керуючий об'єкт, так і керуючий метод керованого об'єкта. Деякі варіанти

1. Монітор здійснює переключення в новий стан **S**, визначає об'єкт, якому необхідно передати керування, і передає вектор станів **S** як параметр відповідному методу-монітору з множини виходів **Y**, а цей метод уже розпізнає і здійснює керування у відповідності зі значенням **S**.

2. Монітор здійснює переключення в новий стан **S**, а потім здійснює вибір об'єкта за значенням **S**, а метод-монітор цього об'єкта здійснює вибір методу за значенням **X**.

3. Монітор здійснює переключення в новий стан **S**, а потім здійснює вибір об'єкта і методу за значенням **S**.

## Література


1. Бадд Т. Объектно-ориентированное программирование в действии / пер. с англ. – С.-Пб.:Питер, 1997. – 464 с. ил.

2. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / пер. с англ. – 2-ое изд. – М.: Бином. – С.-Пб.: Невский диалект, 1998. – 560 с., ил.

3. Шлеер С., Мэллор С. Объектно-ориентированный анализ: моделирование мира в состояниях пер. с англ. – К.: Диалектика, 1993. – 240 с. ил.