

Об'єкти, їх властивості та методи

По-перше, **об'єкт** – це завжди дещо конкретніше, оскільки ми можемо певно визначити його межі, де він починається та де закінчується, що належить до нього, а що ні.

По-друге, **об'єкт завжди має певну внутрішню будову**, про те вона не завжди нам відома.

По-третє, об'єкт характеризується **певною поведінкою**, за якою можна або просто спостерігати (як за польотом НЛО), або навіть впливати на неї (наприклад, змінювати траєкторію польоту НЛО).

Ви самі можете навести багато прикладів різноманітних об'єктів, внутрішня будова яких вам невідома або відома досить поверхньо, проте ви постійно використовуєте їх у своєму житті (холодильник, СВЧ-піч, плита, телевізор, цифровий плеєр, мобільний телефон, фотоапарат і відеокамера тощо). Усі ці об'єкти різні, тож кожний з них має певні **властивості**, які дають змогу відрізнити один об'єкт від іншого. І досить часто нам доводиться оцінювати ці властивості, щоб зрозуміти, які дії та як можна виконувати з об'єктами. Наприклад, якщо ви допомагаєте татові повісити карниз у кімнаті, ви маєте оцінити властивості стінки, до якої будете кріпити карниз, і властивості шурупів, адже стінка може виявитися надзвичайно міцною. Тож треба обрати міцні металеві шурупи саме для неї, а не використовувати маленькі цвяшки, призначені для кріплення на дерев'яній стінці. **Властивості невід'ємно пов'язані з об'єктами, тому вони не можуть існувати без об'єктів.**

Для перевірки та змінювання властивостей об'єкта використовують спеціальні процедури - **методи**, які теж входять до складу об'єкта.

Якщо ми бажаємо працювати з об'єктами, це не означає, що всі об'єкти нам треба створювати самостійно. Об'єкт, створений в одній програмі, можна використовувати в інших. При цьому додавання такого об'єкта не порушує роботу всіх інших об'єктів, використаних у програмі, а поведінка даного об'єкта також не змінюється внаслідок існування інших об'єктів.

Програмні об'єкти можуть реагувати на певні події, що ви бачили неодноразово, працюючи з різноманітним програмним забезпеченням. При виникненні події автоматично запускається спеціальний метод – **обробник даної події**.

За допомогою подій забезпечується взаємодія програми з користувачем. Коли ви переміщуєте мишу або натискаєте її кнопку, це реєструється елементом керування як певна подія, а потрібні дії виконуються в методі-обробнику. Тому, щоб програма могла виконувати те, що бажає користувач, вона має постійно перевіряти стан своїх елементів керування, оскільки конкретні події відповідають типовим прийомам керування. Наприклад, кнопка може реагувати на натискання клавішею миші, крім того, більшість програм здатні розрізняти одинарне та подвійне натискання клавіш миші, використовуючи у цих випадках різні обробники подій.

Основною одиницею в об'єктно-орієнтовному програмуванні є об'єкт, який об'єднує дані з кодом, призначеним для їх обробки, та має:

- *певне ім'я;*
- *властивості*, тобто його характеристики, які можна перевірити або змінити;
- *методи*, тобто дії, які можна виконати над об'єктом;
- *події*, тобто можливі для даного об'єкта ситуації (зміни деяких станів об'єкту), на які він може відповісти заздалегідь визначеними подіями.

Класи об'єктів є шаблонами, які визначають набори властивостей, методів та подій, за якими створюють об'єкти. Об'єкт, що створений за шаблоном класу об'єктів, *є екземпляром класу*, має власне унікальне для даного класу ім'я, проте *наслідує* весь набір властивостей, методів та подій даного класу. Різні екземпляри класу мають однаковий набір властивостей, проте значення властивостей у них мають бути різними.

Кожний об'єкт має певний набір властивостей, які можна змінювати в таких ситуаціях:

- 1) призначити початкові значення під час початкової розробки проекту;
- 2) у режимі виконання проекту з використанням програмного коду.

Щоб об'єкт виконав певну очікувану від нього операцію, потрібно застосувати метод, який він має. Багато з методів об'єктів мають аргументи, які дозволяють призначити параметри виконання об'єктом дії.

Якщо привести аналогію з нашою звичайною мовою, об'єкту відповідає іменник, його властивостям – прикметник, а методи можна зіставити з дієсловом.

При використанні об'єктно-орієнтовного програмування записувати програму у вигляді набору послідовно виконуваних команд незручно, тому використовують *подієвий механізм керування*, за використання якого обробники викликаються при виникненні кожної певної події.

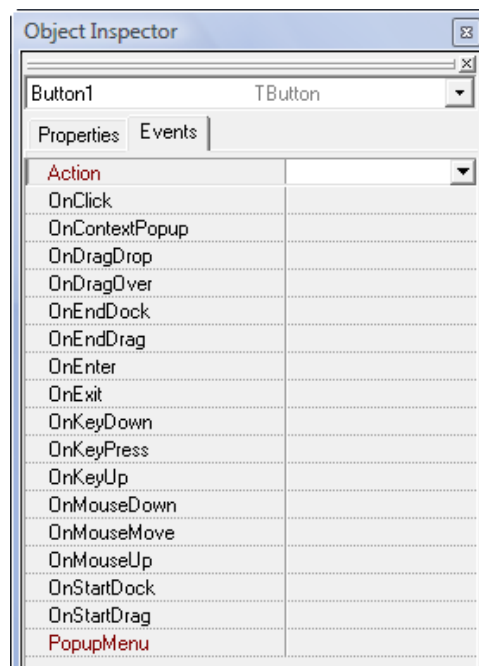
Події та їх обробники

Працюючи з різноманітними програмами, ви звикли до того, що вибір кнопки завжди приводив до настання певної події: відкриття або закриття певного вікна, змінення значень властивостей певних об'єктів, переміщення певного об'єкта по екрану тощо. Але якщо вибрати кнопку в будь-якому з проектів, які ви створювали під час вивчення попереднього уроку, то нічого не відбудеться.

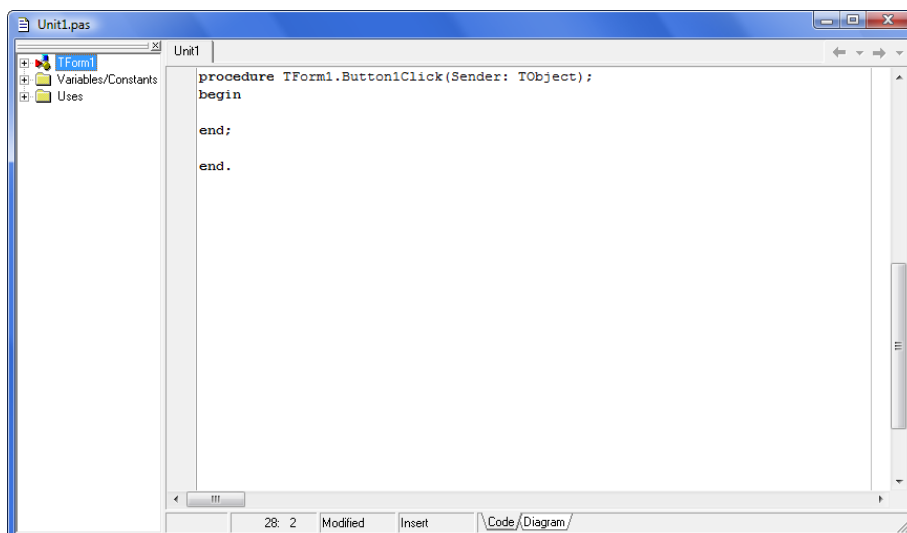
Причиною цієї ситуації є те, що лише самого розміщення на формі кнопки недостатньо, щоб за її вибору щось відбулося. Програмі потрібно «повідомити», яка нова подія повинна відбутися за вибору кнопки.

Якщо виділити кнопку, то у вікні Object Inspector на вкладці Events (англ. events – події) можна вказати, яка подія повинна відбутися як реакція на настання іншої події, наприклад вибір кнопки (рис. 1).

Для цього потрібно двічі клацнути в полі справа від напису **OnClick** (англ. on click – на клацання кнопкою миші). У результаті цих дій у полі **OnClick** на вкладці **Events** з'являється текст **Button1Click** (Button1 – це ім'я виділеної кнопки), а у центральній частині вікна середовища розробки відкривається вікно коду Unit1.pas, на якій створюється заготовка процедури **TForm1.Button1Click**, команди якої й будуть виконуватися після вибору кнопки Button1 (мал. 2)



Мал. 1



Мал. 2

Процедура (лат. procedure – просуватися, йти вперед) – це частина програми, яка має ім'я та яку можна за цим іменем викликати на виконання в різних частинах програми.

Отже, після вибору кнопки **Button1** (подія Click) настає подія **OnClick**, яка полягає у виконанні команд процедури TForm1.Button1Click.

Процедури в Delphi є одним з видів підпрограм. Іншим видом підпрограм у Delphi є функції. Їх розглянемо пізніше.

Процедура, яка виконується при настанні певної події, називається **обробником цієї події**.

Процедура, яка пов'язана з певним об'єктом, називається **методом цього об'єкта**.

Так, процедура **TForm1.Button1Click** є обробником події **OnClick**, яка настає після вибору кнопки **Button1**, а також є методом об'єкта «Кнопка Button1».

Аналогічно викладеному вище можна створювати обробники інших подій, наприклад:

- **OnMouseMove** (англ. on mouse move – на переміщення миші) – ця подія настає після наведення вказівника на кнопку;
- **OnKeyPress** (англ. on key press – на натиснення клавіші) – ця подія настає після натиснення клавіші клавіатури;
- **OnStartDrag** (англ. on start drag – на початок перетягування) – ця подія настає після початку перетягування об'єкта.

Аналогічно можна створювати методи й інших об'єктів, наприклад форми. Список подій на вкладці Events для форми містить події, які ми вже бачили на вкладці для кнопки, а також деякі інші події.

Наприклад:

- **OnCreate** (англ. on create – на створення) – ця подія настає після початку створення форми; команди обробника цієї події виконуються під час створення форми, перед її відкриттям;
- **OnDblClick** (англ. on double click – на подвійне клацання) – ця подія настає після подвійного клацання на формі.

Створення процедур-обробників подій у Delphi

Розглянемо детальніше структуру процедури TForm1.Button1Click – обробника подіїOnClick:

```
procedure TForm1.Button1Click (Sender: TObject);  
begin  
end;
```

Перший рядок процедури – **рядок заголовка**. Він складається зі стандартного слова **procedure** (англ. procedure – процедура), імені процедури **TForm1.Button1Click** і круглих дужок, у яких указується, що дана процедура виконуватиметься в результаті настання певної події з певним об'єктом, у даному випадку події Click з кнопкою Button1. Закінчується рядок заголовка крапкою з комою (;).

Команди процедури, які будуть виконуватися при настанні події **OnClick**, записуються між двома стандартними словами **begin** (англ. begin – початок) і **end** (англ. end – кінець). Команди процедури утворюють **тіло процедури**. Слова **begin** і **end** визначають, де починається і де закінчується тіло процедури. Можна сказати, що вони відіграють роль відкриваючої та закриваючої дужки, між якими записується тіло процедури, тому їх називають **операторними дужками**. Після слова **end** повинна стояти крапка з комою (;). Кожна команда процедури також повинна закінчуватися крапкою з комою (;).

Щоб змінити значення властивостей об'єктів під час виконання проекту, потрібно додати до тексту програми **обробник певної події**. Якщо потрібно, щоб у результаті виконання процедури змінилися значення властивостей одного з об'єктів, процедура повинна містити команди встановлення значення властивостей об'єкта.

Загальний вигляд цих команд такий:

```
<ім'я об'єкта>.<ім'я властивості> := <значення або вираз>;
```

Наприклад,

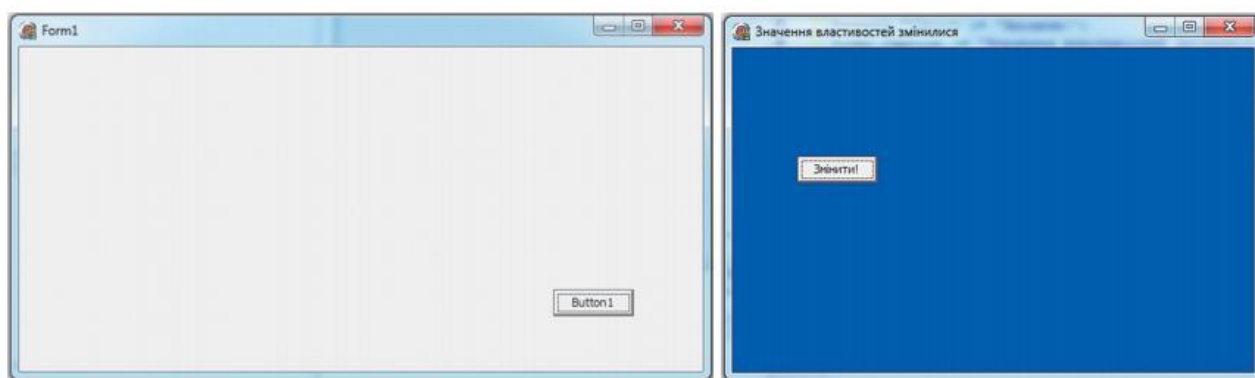
- **Form1.Color := clGreen;** – установити колір фону вікна – зелений;
- **Form1.Left := 300;** – установити відступ лівої межі вікна – 300 пікселів;
- **Form1.Width := Form1.Width*2;** – збільшити поточну ширину вікна вдвічі;
- **Button1.Caption := 'Збільшити';** – установити текст заголовка на кнопці *Збільшити* (якщо значенням властивості є текст, то його потрібно брати в одинарні лапки);
- **Label1.Font.Color := clRed;** – установити колір символів, яким відображатиметься текст у написі, червоний.

Зверніть увагу, що ці команди є різновидами команд присвоювання, які ми вже використовували під час складання алгоритмів.

Наведемо приклад процедури TForm1.Button1Click, виконання якої змінить колір вікна на синій, змінить текст у рядку заголовка, зменшить його ширину на 150 пікселів, змінить положення кнопки і текст на ній:

```
procedure TForm1.Button1Click (Sender: TObject);
begin
  Form1.Color := clBlue;
  Form1.Caption := 'Значення властивостей змінилися';
  Form1.Width := Form1.Width - 150;
  Button1.Left := 60;
  Button1.Top := 100;
  Button1.Caption := 'Змінити!';
end;
```

На **малюнку 3** зліва наведено вигляд вікна після запуску проекту, а справа – після вибору кнопки Button1.

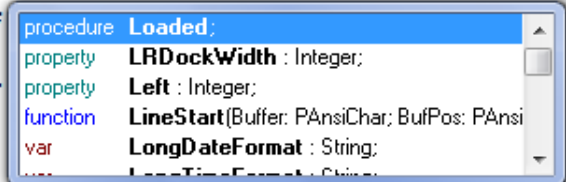


мал. 3

Середовище розробки Delphi 7 має зручні засоби допомоги під час введення тексту проекту:

- після введення першої літери імені об'єкта можна натиснути сполучення клавіш **Ctrl+Пропуск** і відкриється список імен об'єктів, властивостей, процедур, які починаються з цієї літери (мал. 4); з цього списку можна вибрати потрібне і цей текст буде вставлено в текст проекту;

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  L
end;
```



- у ході написання тексту проекту середовище розробки відслідковує синтаксичні правила написання команд;

- якщо ці правила порушуються, то текст з помил-

ками або текст одразу після нього підкреслюється червоною хвилястою лінією;

- потрібно уважно слідкувати за цим і вчасно виправляти всі синтаксичні помилки;
- Delphi 7 не розрізняє великі та малі літери; але якщо слово утворено з кількох слів, прийнято (і зручно) писати першу літеру кожного слова великою, наприклад **TForm1.Button1Click**;
- після введення крапки, що розділяє ім'я об'єкта та ім'я властивості, автоматично відкривається список властивостей даного об'єкта; далі ім'я властивості можна не вводити з клавіатури, а знайти в списку та натиснути клавішу Enter; такий спосіб введення запобігає допущенню синтаксичних помилок, особливо тими, хто не знає англійської мови;
- після запуску проекту на виконання компілятор аналізує текст на наявність синтаксичних помилок; якщо такі помилки будуть знайдені, то після аналізу всіх помилок компіляція проекту переривається та червоним кольором виділяється фон першого з тих рядків тексту проекту, який містить синтаксичні помилки, або наступного за ним рядка;

Ми розглянули, як створити і використати процедуру **TForm1.Button1Click** – обробник події **OnClick**. Аналогічно можна створити інші процедури – **обробники інших подій**.

Уведення та виведення даних під час виконання проекту

Ми розглянули, як змінити значення властивостей елементів керування під час виконання проекту, використовуючи процедуру **TForm1.Button1Click** – обробник події **OnClick**. Але в розглянутому прикладі за кожного вибору кнопки виконуватимуться одні й ті самі команди процедури, а отже, кожного разу однаково змінюватимуться значення вказаних у командах процедури властивостей об'єктів. Якщо необхідно під час виконання проекту кожного разу по-іншому змінювати значення властивостей указаних об'єктів, то потрібно значення цих властивостей якимось чином вводити в програму під час її виконання.

Для **введення даних** під час виконання проекту можна використати поля. Ви вже знаєте, що текст, який знаходиться в полі, є значенням його властивості **Text**. Скористаємося цим.

*Створимо таку процедуру **TForm1.Button1Click** – обробник події **OnClick**, виконання якої дає змогу встановити довільний колір вікна та збільшити його висоту на довільну цілу кількість пікселів.*

Для цього потрібно:

1. Розмістити на формі 2 поля – перше для введення значення кольору вікна, а друге для введення кількості пікселів.
2. Змінити значення властивості **Text** обох цих полів на порожнє.
3. Розмістити зліва від кожного поля напис, змінити значення властивості **Caption** першого на *Колір*, а другого на *Розмір*.
4. Розмістити на формі у правому нижньому куті кнопку, змінити значення її властивості **Caption** на *Змінити*.
5. Створити заготовку процедури **TForm1.Button1Click**, яка виконуватиметься після вибору кнопки.
6. Увести текст процедури **TForm1.Button1Click**:

```
procedure TForm1.Button1Click (Sender: TObject);  
begin  
  Form1.Color := Edit1.Text;  
  Form1.Hight := Form1.Hight + StrToInt (Edit2.Text);  
end;
```

Після запуску проекту відкриється вікно його виконання. У відповідні поля потрібно ввести потрібні значення кольору вікна, наприклад **clRed**, і кількості пікселів для збільшення висоти вікна, наприклад **300**, після чого вибрати кнопку.

Після вибору кнопки настане подія **OnClick**, а отже, запуститься на виконання процедура **TForm1.Button1Click** – обробник події **OnClick**. При виконанні першої команди процедури текст, який буде введено в перше поле і стане значенням властивості **Text** цього поля, буде присвоєний властивості **Color** форми, у результаті чого колір форми стане заданим.

Незважаючи на те, що в друге поле буде введено число, програма сприйматиме його як текст, тобто не зможе виконувати з ним математичних операцій. Тому потрібно спочатку перевести цей текст у ціле число. Для цього використовується функція **StrToInt** (англ. string to integer – рядок у ціле). Після цього це число додається до поточного значення висоти вікна.

Таким чином, при кожному запуску проекту на виконання або перед кожним вибором кнопки можна вводити в поля різні значення, досягаючи тим самим установлення потрібного кольору форми і збільшення його висоти.

Написи можна використовувати не тільки для оформлення підписів об'єктів, а й для виведення результатів виконання проекту.

Створимо **простий калькулятор**, який буде додавати два довільні дійсні числа. Під час виконання проекту користувач вводитиме по одному довільному дійсному числу в кожне поле і вибиратиме кнопку із заголовком

Додати. За вибору кнопки настане подія **OnClick**, в результаті чого виконуватиметься процедура **TForm1.Button1Click**, яка додаватиме ці числа і виводитиме суму в напис.

Для створення такого проекту потрібно:

1. Розмістити на формі два поля, напис для виведення результату, три написи для оформлення та кнопку.
2. Установити порожні значення властивості **Text** двох полів і властивості **Caption** напису для виведення суми.
3. Установити відповідні значення властивості **Caption** для інших трьох написів і кнопки.
4. Створити заготовку процедури **TForm1.Button1Click**, яка виконуватиметься після вибору кнопки.
5. Увести текст процедури **TForm1.Button1Click**. Розглянемо призначення кожної команди наведеної процедури.

```
procedure TForm1.Button1Click(Sender: TObject);  
var x, y, z: Real;  
begin  
    x := StrToFloat(Edit1.Text);  
    y := StrToFloat (Edit2.Text);  
    z := x+y;  
    Label1.Font.Color := clRed;  
    Label1.Caption := FloatToStr(z);  
end;
```

Як ми вже зазначали вище, число, яке вводиться в поле, стає значенням його властивості **Text** і сприймається програмою як текст.

Тому перші дві команди процедури є **командами присвоювання**, які призначені для **переведення текстового представлення дійсних чисел у самі дійсні числа** (для цього використана функція **StrToFloat** (англ. string to float – рядок у число з плаваючою десятковою крапкою, тобто у дійсне число) і присвоювання цих чисел змінним **x** та **y** відповідно.

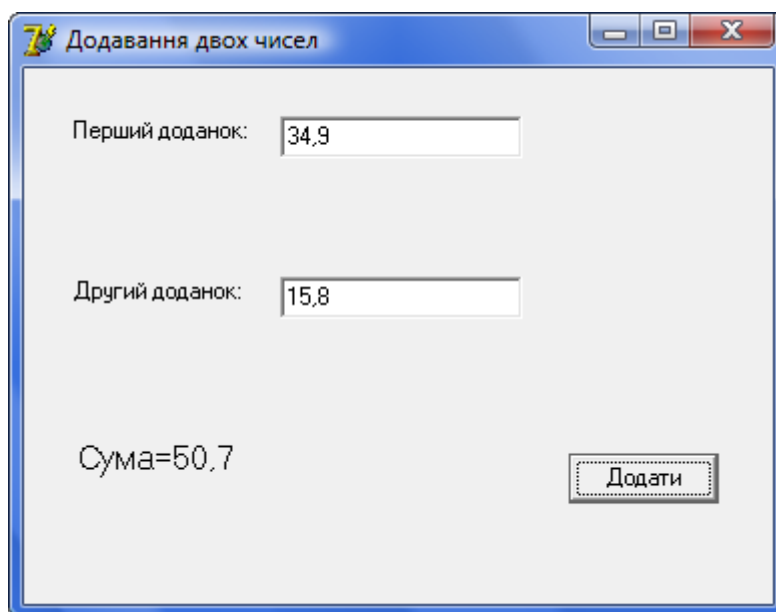
Третя команда додає ці числа і присвоює результат змінній **z**.

Четверта команда встановлює червоним колір символів, яким виводиться результат у напис.

П'ята команда переводить результат додавання (значення змінної **z**) із числа в текст, використовуючи функцію **FloatToStr** (англ. float to string – число з

плаваючою десятковою крапкою у рядок) і присвоює цей текст значенню властивості **Caption** напису з іменем **Label1** для виведення результату.

Під час виконання створеного проекту користувач уводить у поля числа (наприклад, $-34,9$ і $15,8$), вибирає кнопку **Додати** і одержує результат $-50,7$.



Не закриваючи проект, користувач може ввести в поля інші числа, знову вибрати кнопку **Додати** й одержати новий результат. І так можна повторювати довільну кількість разів. Це демонструє таку властивість алгоритму як **масовість**.

Іноді зручно виводити повідомлення не в написи на формі, а в **спеціальне окреме вікно**. Це можна здійснити, включивши до тексту програми команду **ShowMessage(<текст>)** (англ. **show message** – показати повідомлення). За цієї команди відкривається спеціальне **вікно виведення повідомлення**, в якому виводиться текст, указаний у команді. Якщо таку команду включити до процедури, що розглядалася вище, для виведення суми двох уведених дійсних чисел замість останньої команди виведення результату в напис або після цієї команди, то вона виглядатиме так:

```
ShowMessage('Сума '+FloatToStr(z));
```

Знак **плюс** у дужках означає **з'єднання двох указаних текстів**. Пропуск після слова **Сума** використовують для того, щоб у вікні відокремити число від тексту.

При використанні вікна повідомлення для завершення виконання проекту спочатку слід закрити вікно повідомлення, а потім – вікно проекту.

