Об'єктно-орієнтоване програмування

на основі мови C++

2й семестр

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning."

Rich Cook

Лекція №1

викладач:

Доцент кафедри «Динаміка і міцність машин»

к.т.н. Водка Олексій Олександрович (к. 12)

курс:

0.5 лекція на тиждень

2.5 л / р в тиждень

Іспит(100 б) = Лаб(60 б) + Тест (40 б)

література

- Шілдт Г. Теорія і практика С ++. - СПб .: BHV, 1996.
- Р. Лафоре. Об'єктно-орієнтоване програмування в С ++. - СПб: ПІТЕР, 2003 г. - 928 с.
- **Павловська Т.О.** С / С ++. Програмування на мові високого рівня. -СПб.: Питер, 2001, 2003.
- **Павловська Т.О., Щупак Ю.А.** С ++. Об'єктно-орієнтоване програмування: Практикум. - СПб: ПІТЕР, 2004.
- Стефан Р. Девіс. С ++для чайників, Вільямс, Діалектика, 2001 г.
- Дейтел. Як програмувати на С ++. Біном, 2003м
- Бйорн Страуструп. Мова програмування С ++. Спеціальне видання, Біном - 2006, 1104 с.

Введення в об'єктно-орієнтоване програмування (ООП)

Конструктори і деструктори

процедурні мови

- Розвиток об'єктно-орієнтованого методу обумовлено обмеженістю інших методів програмування, розроблених раніше.
- C, Pascal, FORTRAN та інші подібні з ними мови програмування відносяться до категорії процедурних мов. Кожен оператор такої мови є вказівкою до комп'ютера зробити деяку дію, наприклад прийняти дані від користувача, зробити з ними певні дії і вивести результат цих дій на екран. Програми, написані на процедурнихмовах, Являють собою послідовності інструкцій.

процедурні мови

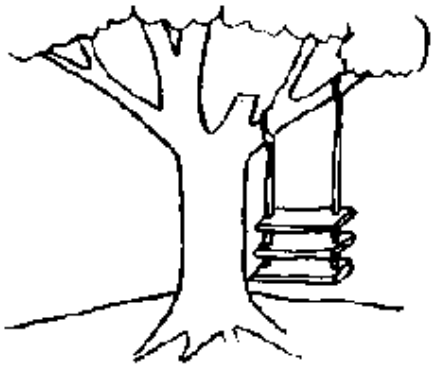
Дані + Алгоритм = Програма

Розподіл на функції

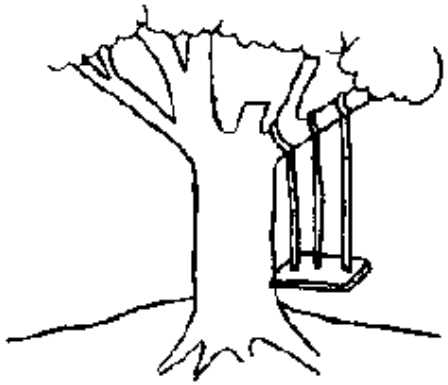
- Коли розмір програми великий, список команд стає занадто громіздким. Дуже невелика кількість програмістів здатне утримувати вголові більш 500 рядків програмного коду, якщо цей код не розділений па більш дрібні логічні частини. функція є засобом, що полегшує сприйняття при читанні тексту програми.
- Програма, побудована на основі процедурного методу, розділена на функції, кожна з яких в ідеальному випадку виконує деяку закінчену послідовність дій і має явно виражені зв'язки з іншими функціями програми

Недоліки структурного програмування

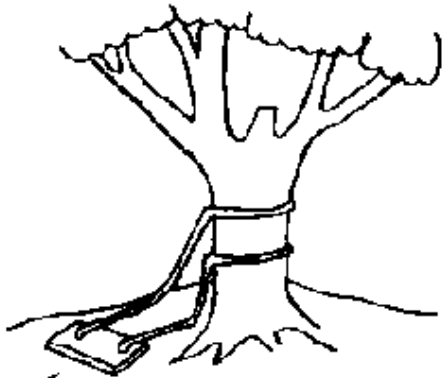
- У безперервному процесі зростання і ускладнення програм стали поступово виявлятися недоліки структурного підходу до програмування.
- Робота над програмним проектом відбувається за наступним принципом: задача виявляється складніше, ніж здавалося, терміни здачі проекту переносяться. Усе нові й нові програмісти залучаються для роботи, що різкозбільшує витрати. Завершення роботи знову переноситься, і в результатіпроект терпить крах.



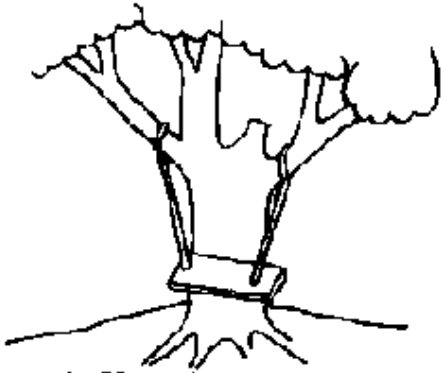
1. Как было предложено организатором разработки



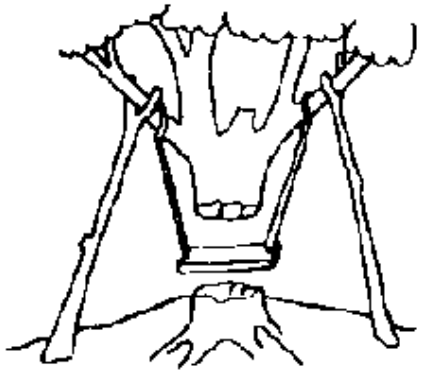
2. Как было описано в техническом задании



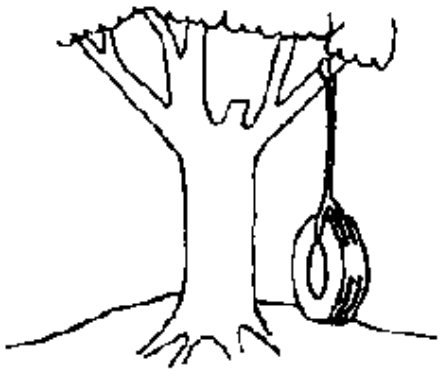
3. Как было спроектировано ведущим системным специалистом



4. Как было реализовано программистами



5. Как было внедрено



6. Чего хотел пользователь

Два нестачі процедурного програмування

- необмеженість доступу функцій до глобальним даними.
- поділ цих та функцій, що є основою структурного підходу, погано відображає картину реального світу.

- У процедурній програмі, написаної, наприклад, на мові С, існує два типи даних.
- локальні дані знаходяться всередині функції і призначені для використання виключно цією функцією і не можуть бути змінені іншими функціями.
- Якщо існує необхідність спільного використання одних і тих ж даних декількома функціями, то дані повинні бути оголошені як глобальні.

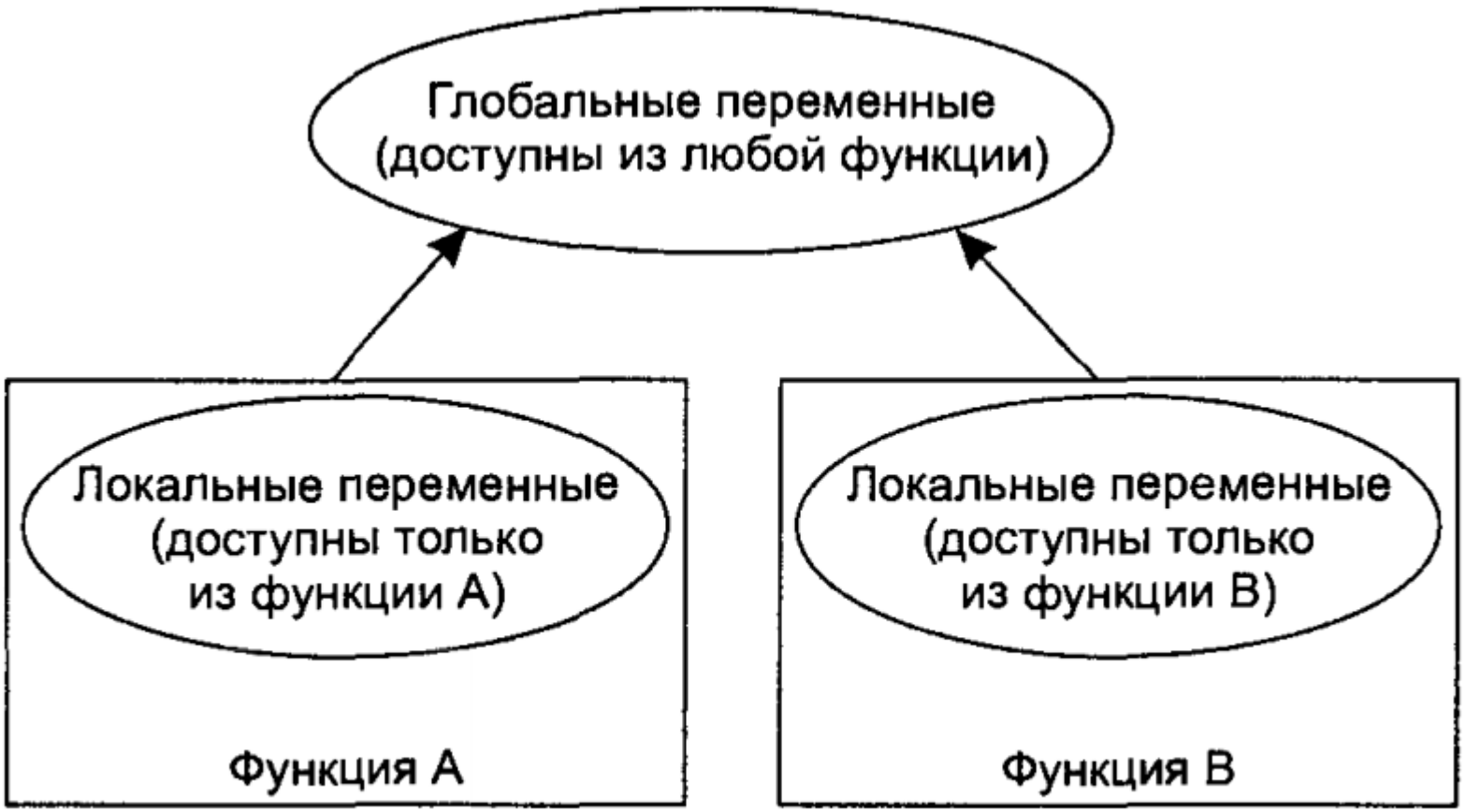


Рис. 1.1. Глобальные и локальные переменные

- Великі програми зазвичай містять безліч функцій і глобальних змінних. Проблема процедурного підходу полягає в тому, що числоможливих зв'язків між глобальними змінними і функціями може бути дуже велике.

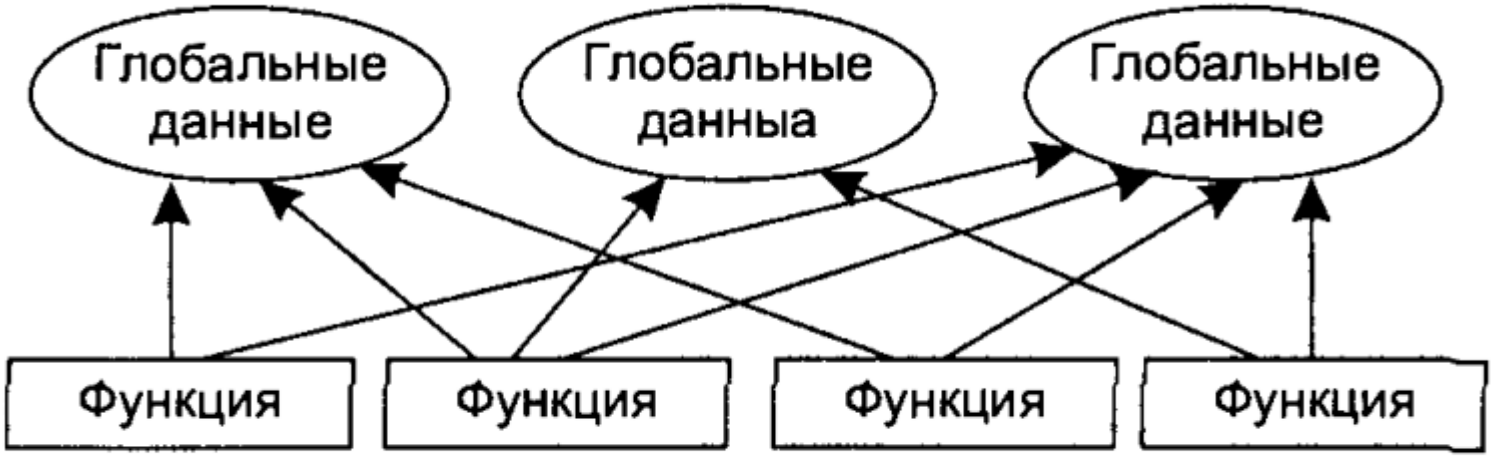


Рис. 1.2. Процедурный подход

Велике число зв'язків між функціями і даними породжує кілька проблем:

- ускладнюється структура програми
- В програму стає важко вносити зміни.

Наприклад, якщо розробник програми складського обліку вирішить зробити код продукту не 5-значним, а 12-значним, то буде необхідно змінити відповідний тип даних з short на long. Це означає, Що в усі функції, які оперують кодом продукту, повинні бути внесені зміни, що дозволяють обробляти дані типу long. Таким чином, будь-яка зміна тягне за собою далекосяжні наслідки.

Моделювання реального світу

Друга, більш важлива, проблема процедурного підходу полягає в тому, що відділення даних від функцій виявляється малопридатним для відображення картини реального світу. У реальному світі нам доводиться мати справу зфізичними об'єктами, такими, наприклад, як люди або машини. ці об'єктине можна віднести ні до даних, ні до функцій, оскільки реальні речі являють собою сукупність властивостей і поведінки.

властивості

Прикладами властивостей (іноді званих характеристиками) для людей можуть бути колір очей або місце роботи; для машин - потужність двигуна ікількість дверей. Таким чином, властивості об'єктів рівносильні даними впрограмах: Вони мають певне значення, наприклад блакитний для кольору очей або 4 для кількості дверей автомобіля.

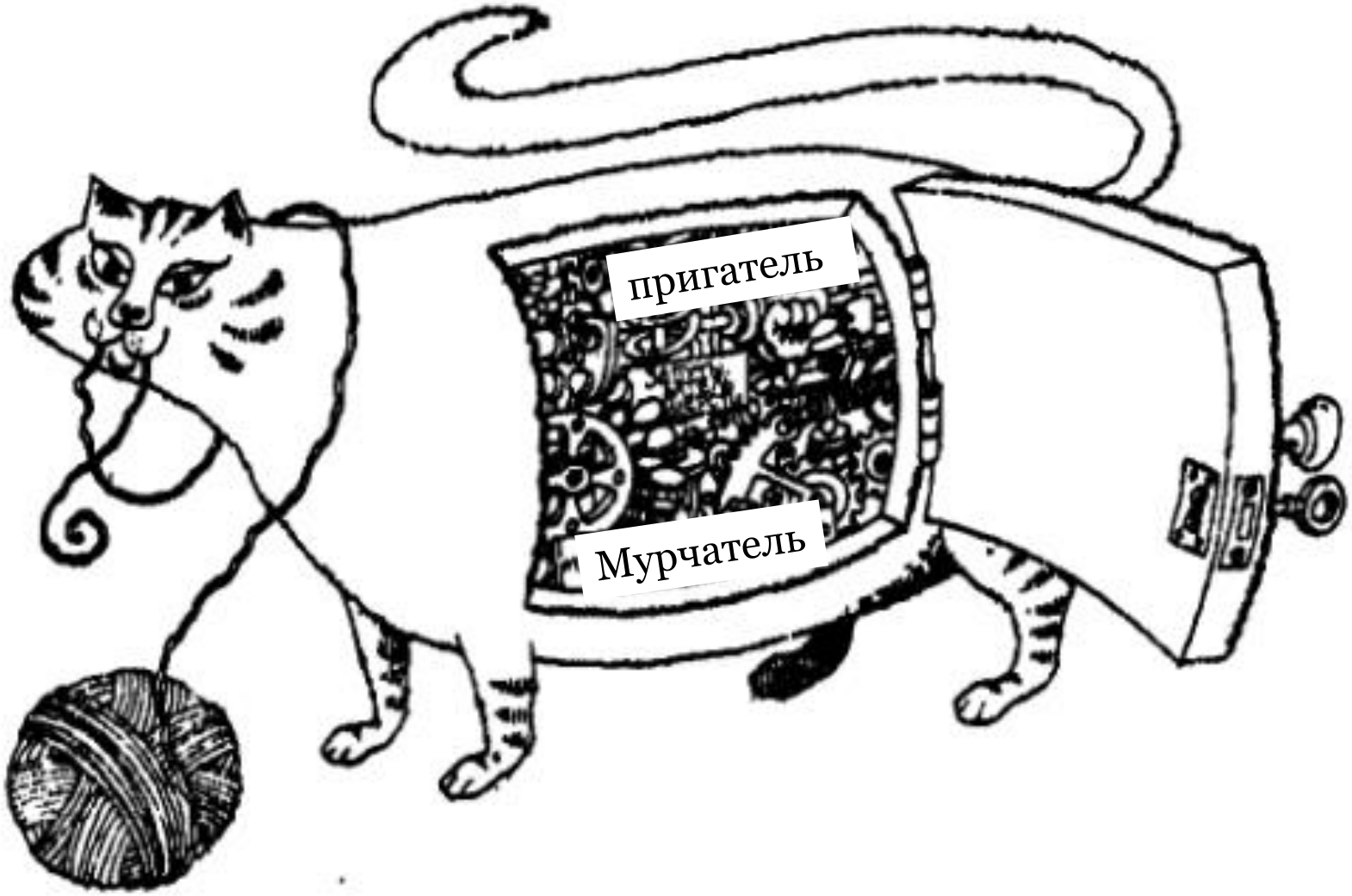
Поведінка

Поведінка - це деяка реакція об'єкта у відповідь на зовнішній вплив. наприклад, Ваш бос у відповідь на прохання про підвищення може дати відповідь «так» або «ні». Якщо ви натиснете на гальмо автомобіля, це спричинить за собою йогозупинку. Відповідь і зупинка є приклади поведінки. поведінка схожез функцією: ви викликаєте функцію, щоб здійснити будь-яку дію (на приклад, Вивести на екран обліковий запис), і функція здійснює цю дію. Таким чином, ні окремо взяті дані, ні окремо взяті функціїне здатні адекватно відобразити об'єкти реального світу.

Об'єктно-орієнтований підхід

- Основною ідеєю об'єктно-орієнтованого підходи є об'єднання даних і дій, вироблених над тими даними, в єдине ціле, яке називається об'єктом.
- Функції об'єкта, звані в С ++ методами або функціями-членами, зазвичай призначені для доступу до даних об'єкта. Якщо необхідновважати будь-які дані об'єкта, потрібно викликати відповідний метод, Котрий виконає зчитування і поверне потрібну установку. прямий доступ до даних неможливий.

інкапсуляція- об'єднання даних і методів для роботи з ними в один об'єкт. Інкапсуляція також реалізує приховування даних від зовнішнього впливу, що захищаєїх від випадкового зміни.

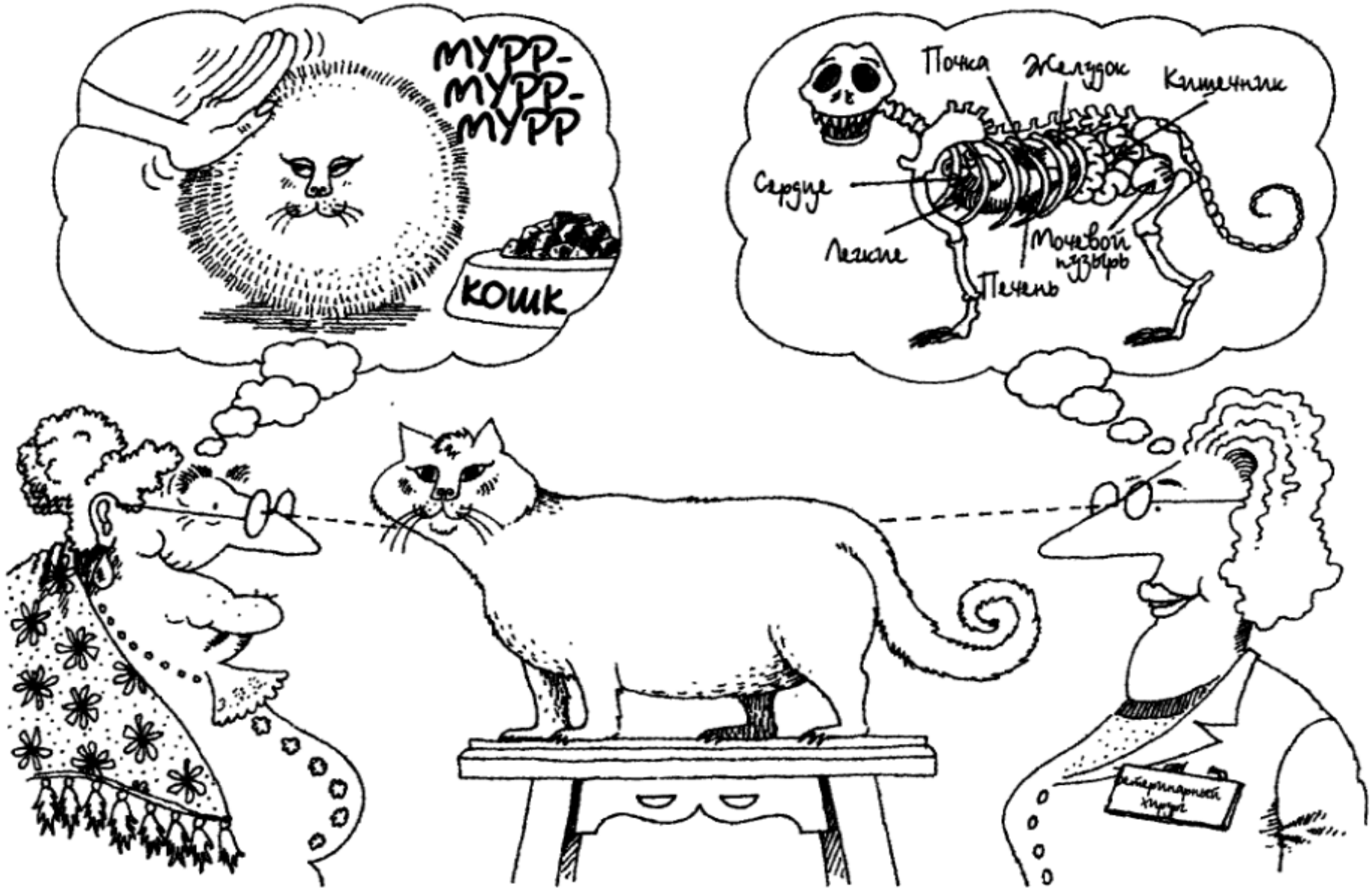


Инкапсуляция скрывает детали реализации объекта

абстрагування

- абстрагування - один з основних методів, що дозволяють впоратися із складністю.

абстракція виділяє істотні характеристики деякого об'єкта, Що відрізняють його від всіх інших видів об'єктів і таким чином, Чітко описує його концептуальні межі з точки зору спостерігача.



Абстракция концентрирует внимание на существенных свойствах объекта с точки зрения наблюдателя

Об'єктно-орієнтоване програмування ООП (object-oriented programming - OOP) - це метод програмування, Заснований на представленні програми у вигляді сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії успадкування.

Моделі реальних об'єктів (абстракції) + поведінку об'єктів (взаємодія) = програма

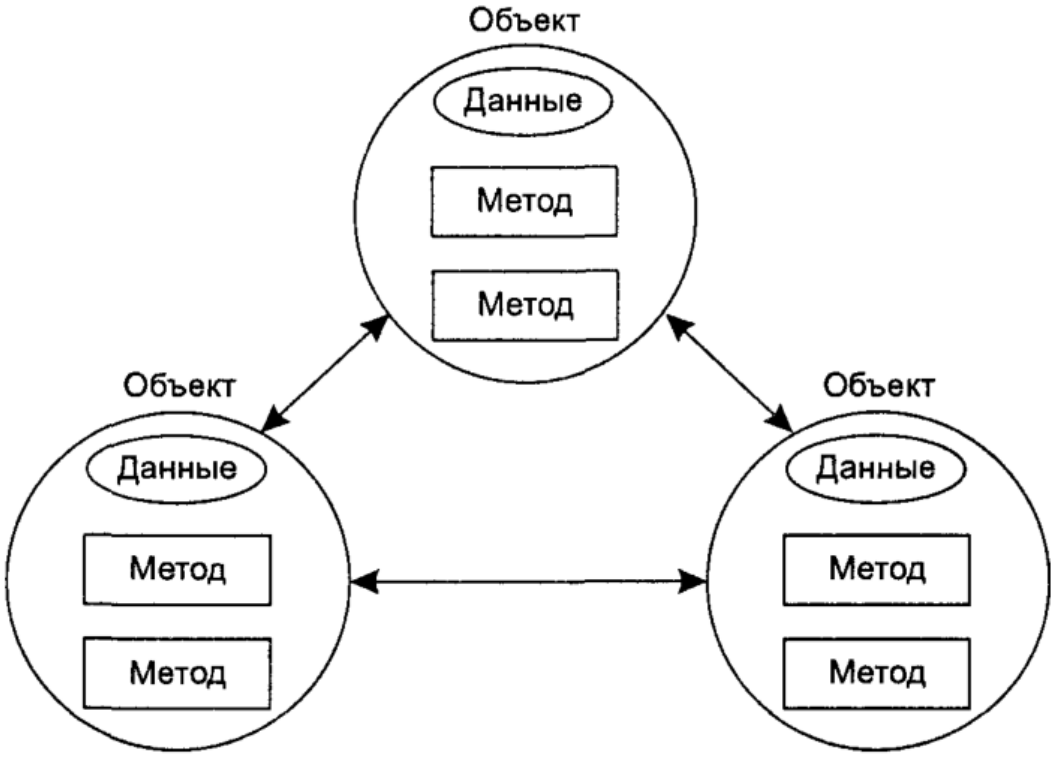


Рис. 1.3. Объектно-ориентированный подход

Що може бути об'єктом (класом)?

Фізичні об'єкти.

- Автомобілі під час моделювання вуличного рухи.
- схемні елементи при моделюванні ланцюга електричного струму.
- країни при створенні економічної моделі.
- літаки при моделюванні диспетчерської системи.

елементи інтерфейсу

- вікна.
- Меню.
- графічні об`єкти (лінії, прямокутники, кола).
- миша, Клавіатура, дискові пристрої, принтери.

структури даних

- масиви.
- стеки.
- пов'язані списки.
- бінарні дерева.

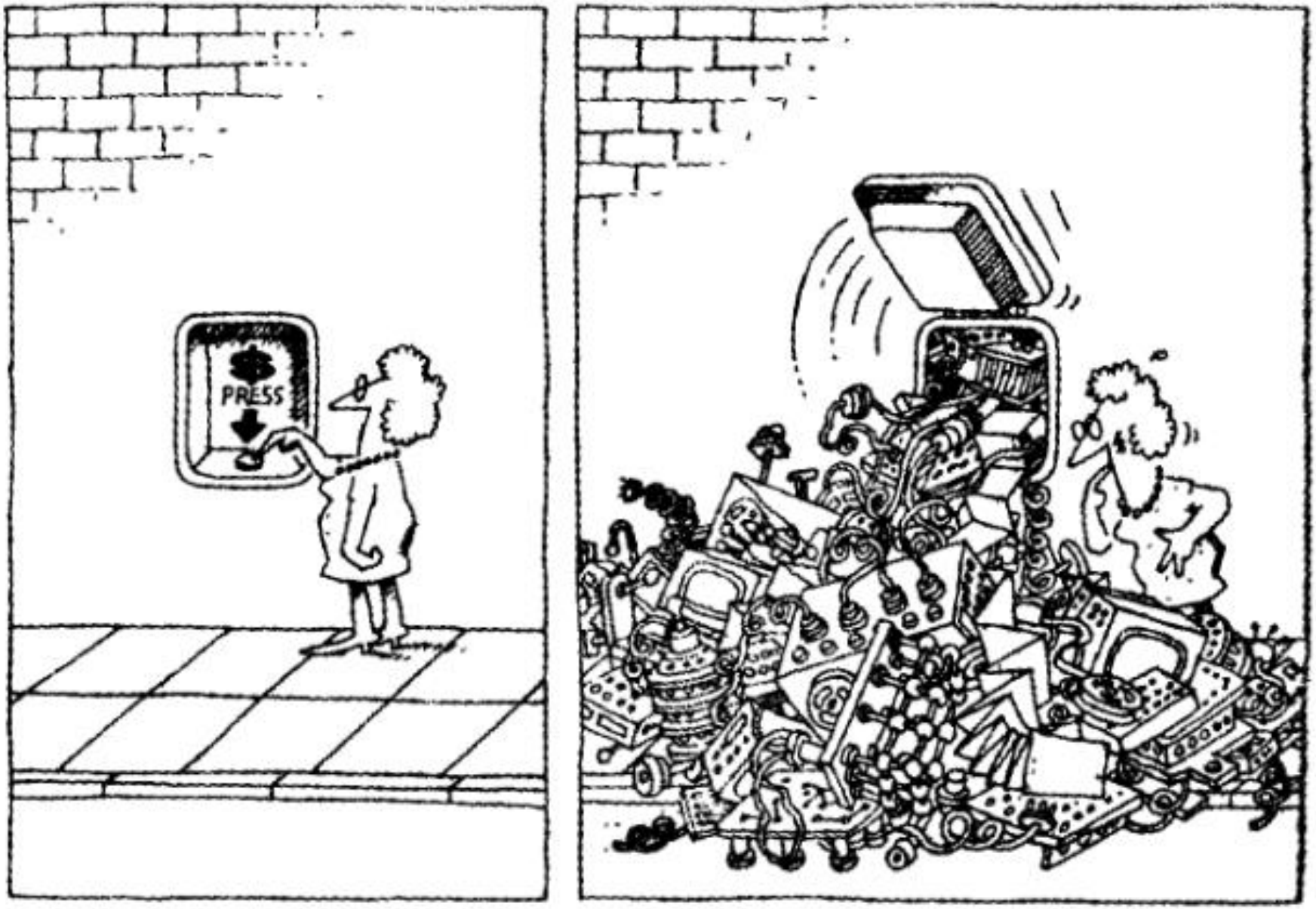
Групи людей
- Співробітники.
- Студенти.
- Покупці.
- Продавці.

сховища даних
- Описи інвентарю.
- Списки співробітників.
- Словники.
- Географічні координати міст світу.

призначені для користувача типи даних
- Час.
- Величини кутів.
- Комплексні числа.
- Точки на площині.

Учасники комп'ютерних ігр
- Автомобілі в гонках.
- Позиції в настільних іграх (шашки, шахи).
- Тварини в іграх, пов'язаних з живою природою.
- Друзі та вороги в пригодницьких іграх.

Та інше



Задача группы проектировщиков — создать иллюзию простоты

клас

- Клас є абстрактним типом даних, що визначаються користувачем, і являє собою модель реального об'єкта у вигляді даних і функцій для роботи з ними.
- Дані класу називаються полями (по аналогії з полями структури), а функції класу - методами. Поля і методи називаються елементами класу. опис класу в першому наближенні виглядає так:

```
class <Ім'я> {
[ private:]
<Опис прихованих елементів>
public:
<Опис доступних елементів>
}; // Опис закінчується крапкою з комою
```

private і public

- специфікатори доступу **private** і **public** керують видимістю елементів класу. Елементи, описані після службового словаprivate, Видимі тільки всередині класу. Цей вид доступу прийнятий в класі за замовчуванням.
- Інтерфейс класу описується після специфікатора public. Дія будь-якого специфікатор поширюється до наступного специфікатора або до кінця класу. Можна, можливозадавати кілька секцій private і public, Порядок їх слідування значення не має.

Приклад опису класу

```
class monstr{
 int health, ammo; //поля класу
 char * Name;
 public: // специфікатор доступу
 // конструктор
 monstr(int he = 100, int am = 10)
 { health = he; ammo = am;}
 void draw (int x, int y,
 int scale, int position);
 int get_health() {Return health;}
 int get_ammo() {Return ammo;}
};
```

void monstr:: draw (int x, int y, int scale, int position) {/ * Тіло методу * /}

inline int monstr::get_ammo() {return ammo;}

опис об'єктів

```
  monstr Vasia;
  monstr Super(200, 300);
  monstr stado[100];
  monstr *beavis = new monstr (10);
monstr &butthead = Vasia;
```

Доступ до елементів об'єкта

```
int n = Vasia.get_ammo();
stado[5] .draw;
cout << beavis->get_health();
```

конструктори

Конструктор - метод класу, який служить для створення і ініціалізації екземпляра класу. Ім'я конструктора завжди збігається з ім'ям класу.

- Конструктор не повертає значення, навіть типу void. Можна отримати покажчик на конструктор.
- Клас може мати кілька конструкторів з різними параметрами для різних видів ініціалізації (при цьому використовується механізм перевантаження).
- Конструктор, викликаний без властивостей, називається конструктором за замовчуванням.
- Параметри конструктора можуть мати будь-який тип, крім цього ж класу. Можна задавати значення параметрів за замовчуванням. Їх може містити тільки один з конструкторів.

деструктори

Деструкція - метод класу, який служить для видалення екземпляра класу. Ім'я деструктора складається з символу ~ (тильда) і імені класу. деструкція НЕ має аргументів і повертається значення.

- Деструкція викликається автоматично, коли об'єкт виходить з області видимості:
- для локальних об'єктів - при виході з блоку, в якому вони оголошені;
- для глобальних - як частина процедури виходу з main;
- для об'єктів, заданих через покажчики, деструктор викликається неявно при використанні операції delete.

Деструкція для розглянутого прикладу повинен виглядати так

```
monstr:: ~monstr() {Delete [] name;}
```

Відділення інтерфейсу від реалізації

Один з найбільш фундаментальних принципів розробки хорошого програмного забезпечення полягає в відділенні інтерфейсу від реалізації:

при побудові програми на C ++ кожне визначення класу зазвичай поміщається в заголовки, а визначення методів цього класу поміщаються в файли вихідних кодів з тими ж базовими іменами.

*Заголовки - **ім'я_класса.h***

*Файл вихідного коду - **ім'я_класса.cpp***

Заголовки включаються (через #include) в кожен файл, в якому використовується клас, а файли з вихідними кодами компілюються і компонуються з файлом, що містить головну програму (main).

***Приклад.** Порядок викликів конструкторів і деструкторів.*

```
// Файл CreateAndDestroy.h
class CreateAndDestroy
{
public:
 CreateAndDestroy(int); //конструктор
 ~CreateAndDestroy(); // деструкція
private:
 int data;
};
// -----------------Кінець файлу -----

//файл CreateAndDestroy.cpp
#include <iostream>
using namespace std;
CreateAndDestroy::CreateAndDestroy(int value)
{
 data = Value;
 cout << "Object" << data << "constructor" << endl;
}
CreateAndDestroy:: ~CreateAndDestroy()
{ cout << "Object" << data << "destructor" << endl;
}
// ------- Кінець файлу------------
```

```
// Файл main.cpp
#include <iostream>
#include "CreateAndDestroy.h"
using namespace std;
CreateAndDestroy first (1);
int main ()
{ cout << "main: Hello" << endl;
 CreateAndDestroy second (2);
 cout << "main: second created" << endl;
 CreateAndDestroy* third = new CreateAndDestroy(3);
 cout << "main: third created" << endl;
 delete third;
 cout << "main: third deleted» << endl;
 cout << "Exit from main" << endl;
 return 0;
}
```

```
Object 1 constructor
main: Hello
Object 2 constructor
main: second created
Object 3 constructor
main: third created
Object 3 destructor
main: third deleted
Exit from main
Object 2 destructor
Object 1 destructor
```