Задача. Дано ієрархію класів Object ◄— NamedObject, існує контейнер вказівників на об’єкти базового класу для виклику віртуальних методів, що реалізують поліморфну поведінку. Слід додати код, що надає можливість відображення та редагування об’єктів з урахуванням їх специфіки (не через визначені у базовому класі методи) без внесення суттєвих змін до реалізації Object та NamedObject. **Варіант №1**: додаємо поле «тип» та аналізуємо його через “switch case”. Недоліки: “switch case” у багатьох місцях програми, вноситься додаткове поле «тип».

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

////////////////////////////////////////////////////////////////////////////

enum Type { OBJECT, NAMED_OBJECT };
#define TYPE(NAME) virtual Type getType() const { return NAME ; }

////////////////////////////////////////////////////////////////////////////

class Object
{
public:
    virtual ~Object() {}
    virtual string say()
    {
        return "I'm Object";
    }
    TYPE (OBJECT)
};

class NamedObject: public Object
{
    string n;
public:
    NamedObject(const string& name = "John Doe") : n(name) {}
    string say()
    {
        return "My name is " + n;
    }
    void setName(const string & name)
    {
        n = name;
    }
    TYPE (NAMED_OBJECT)
};

////////////////////////////////////////////////////////////////////////////

void display(Object& obj) { cout << "Object say: " << obj.say() << endl; }
void manage(Object& obj) { cout << "There is nothing to do with me" << endl; }

void manage(NamedObject& obj)
{
    cout << "Enter new name: " << endl;
    string n;
    cin >> n;
    obj.setName(n);
}

////////////////////////////////////////////////////////////////////////////

int main()
{
    vector<Object*> objects;
    objects.push_back(new Object());
    objects.push_back(new NamedObject());

    for(unsigned i;;)
    {
        cout << "enter index of object to work with:";
        cin >> i;
        if (i >= objects.size()) break;
        cout << "Direct call: " << objects[i]->say() << endl;
        switch (objects[i]->getType())
        {
            case OBJECT:
                            display(*objects[i]);
                            manage(*objects[i]);
                 break;
            case NAMED_OBJECT:
                            display((NamedObject&)*objects[i]);
                            manage((NamedObject&)*objects[i]);
                 break;
        }
    }
}
```

Варіант №2: використовуємо динамічну інформацію про тип. Недоліки: “сходинки IF ELSE” у багатьох місцях програми.

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

////////////////////////////////////////////////////////////////////////////

class Object
{
public:
    virtual ~Object() {}
    virtual string say()
    {
        return "I'm Object";
    }
};

class NamedObject: public Object
{
    string n;
public:
    NamedObject(const string& name = "John Doe") : n(name) {}

    string say()
    {
        return "My name is " + n;
    }
    void setName(const string & name)
    {
       n = name;
    }
};

////////////////////////////////////////////////////////////////////////////

void display(Object& obj) { cout << "Object say: " << obj.say() << endl; }
void manage(Object& obj) { cout << "There is nothing to do with me" << endl; }
void manage(NamedObject& obj)
{
    cout << "Enter new name: " << endl;
    string n;
    cin >> n;
    obj.setName(n);
}

////////////////////////////////////////////////////////////////////////////

int main()
{
    vector<Object*> objects;
    objects.push_back(new Object());
    objects.push_back(new NamedObject());

    for(unsigned i;;)
    {
        cout << "enter index of object to work with:";
        cin >> i;
        if (i >= objects.size()) break;
        cout << "Direct call: " << objects[i]->say() << endl;
        if (NamedObject* obj = dynamic_cast < NamedObject* > (objects[i]))
        {
            display(*obj);
            manage(*obj);
        }
        else if (typeid(objects[i]) == typeid(Object*))
        {
            display((Object&)*objects[i]);
            manage((Object&)*objects[i]);
        }
    }
}
```

Варіант №3: використовуємо патерн “Visitor”. Недолік: деяка модифікація Object та NamedObject.

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

////////////////////////////////////////////////////////////////////////////
class Object;
class NamedObject;

class Processor
{
public:
    virtual void operator() (Object&) {};
    virtual void operator() (NamedObject& obj) {};
};
#define PROCESSIBLE virtual void processBy(Processor& oper) { oper(*this); }

////////////////////////////////////////////////////////////////////////////

class Object
{
public:
    virtual ~Object() {}
    virtual string say()
    {
        return "I'm Object";
    }
    PROCESSIBLE
};

class NamedObject: public Object
{
    string n;
public:
    NamedObject(const string& name = "John Doe") : n(name) {}

    string say()
    {
        return "My name is " + n;
    }
    void setName(const string & name)
    {
        n = name;
    }
    PROCESSIBLE
};

////////////////////////////////////////////////////////////////////////////

class Display: public Processor
{
public:
    void operator() (Object& obj) { cout << "Object say: " << obj.say() << endl; }
    void operator() (NamedObject& obj) { (*this)((Object&)obj); }
};

class Manager: public Processor
{
public:
    void operator() (Object& obj) { cout << "There is nothing to do with me" << endl; }
    void operator() (NamedObject& obj)
    {
        cout << "Enter new name: " << endl;
        string n;
        cin >> n;
        obj.setName(n);
    }
};
////////////////////////////////////////////////////////////////////////////
int main()
{
    vector<Object*> objects;
    objects.push_back(new Object());
    objects.push_back(new NamedObject());

    for(unsigned i;;)
    {
        Display display;
        Manager manager;
        cout << "enter index of object to work with:";
        cin >> i;
        if (i >= objects.size()) break;
        cout << "Direct call: " << objects[i]->say() << endl;
        objects[i]->processBy(display);
        objects[i]->processBy(manager);
    }
}
```

Варіант №4: використовуємо ієрархію класів-обгорток (без наслідування від Object).

```
#include <iostream>
#include <vector>
using namespace std;
//////////////////////////////////////////////////////////////////////////////
class Object
{
public:
    virtual ~Object() {}
    virtual string say()
    {
        return "I'm Object";
    }
};

class NamedObject: public Object
{
    string n;
public:
    NamedObject(const string& name = "John Doe") : n(name) {}

    string say()
    {
        return "My name is " + n;
    }
    void setName(const string & name)
    {
        n = name;
    }
};
//////////////////////////////////////////////////////////////////////////////

class ObjectInterface
{
public:
    virtual ~ObjectInterface() {}
    virtual Object& getObject() = 0;
    virtual void display() = 0;
    virtual void manage() = 0;
};

//////////////////////////////////////////////////////////////////////////////

class InteractiveObject: public ObjectInterface
{
    Object obj;
public:
    InteractiveObject(const Object& o) : obj(o) {}
    Object& getObject() { return obj; }
    void display() { cout << "Object say: " << obj.say() << endl; }
    void manage() { cout << "There is nothing to do with me" << endl; }
};

class InteractiveNamedObject: public ObjectInterface
{
    NamedObject obj;
public:
    InteractiveNamedObject(const NamedObject& o) : obj(o) {}
    Object& getObject() { return obj; }
    void display() { cout << "Object say: " << obj.say() << endl; }
    void manage()
    {
        cout << "Enter new name: " << endl;
        string n;
        cin >> n;
        obj.setName(n);
    }
};
//////////////////////////////////////////////////////////////////////////////
int main()
{
    vector<ObjectInterface*> objects;
    objects.push_back(new InteractiveObject(Object()));
    objects.push_back(new InteractiveNamedObject(NamedObject()));

    for(unsigned i;;)
    {
        cout << "enter index of object to work with:";
        cin >> i;
        if (i >= objects.size()) break;
        cout << "Direct call: " << objects[i]->getObject().say() << endl;
        objects[i]->display();
        objects[i]->manage();
    }
}
```