LABORATORUL 6

1 Upcasting

În programarea orintată pe obiecte putem asigna un obiect de tip derivat către o variabilă / referința / pointer de tip bază. Acest fenomen se numește *upcasting*. Acest lucru este permis deoarece clasa derivată are aceleași proprietăți și metode cu clasa de bază, prin urmare un obiect de tip derivat poate substitui orcând un obiect de tip bază.

```
#include <iostream>
class B {
public:
    B () {std::cout << "B";}
};

class D : public B {
public:
    D () {std::cout << "D";}
};

int main () {
    D d;
    B b = d;
    B *pb = &d; // B *pb = new D();
    B &rb = d;
    return 0;
}
```

Atenție! Operația inversă (asignare obiect de tip bază către variabilă / referința / pointer de tip derivat – *downcasting*) nu este întotdeauna posibilă.

2 Metode virtuale

Metodele virtuale sunt metode ale căror funcționalitate poate fi rescrisă în clasa derivată. Spre deosebire de metodele normale, funcționalitate rescrisă se păstrează atunci când un obiect de tip derivat este utilizat folosind referințe și pointeri de tip bază.

Pentru a declara o metodă virtuală trebuie adăugat cuvântul cheie `virtual` înainte de de signatura metodei.

```
#include<iostream>
using namespace std;
class B {
    public:
    virtual void foo() {
        cout << "B";
    }
};
class D: public B {
public:
    void foo () {
        cout << "D";
    }
};

int main() public
```

```
{
    B b = D();
    b.foo();                // B
    B *pb = new D();
    pb->foo();              // D
    return 0;
}
```

Atunci când facem upcasting, la apelarea unei metode virtuale se folosește cea mai recentă redefinire a metodei (nu suntem nevoiți să redefinim la fiecare moștenire);

```
#include<iostream>
using namespace std;
class B {
    public:
    virtual void foo() {
        cout << "B::foo" << endl;
    }
    virtual void bar() {
        cout << "B::bar" << endl;
    }
};
class D: public B {
public:
    void foo () {
        cout << "D::foo" << endl;
    }

    void bar() {
        cout << "D::bar" << endl;
    }
};

class E: public D {
public:
    void bar () {
        cout << "E::bar" << endl;
    }
};

int main()
{
    B *b = new E();
    b->foo();         // D::foo
    b->bar();         // E::bar
    return 0;
}
```

Apelarea corectă a funcților virtuale se realizează prin `*vptr` și `VTABLE` . Când declarăm o metodă virtuală, fiecare obiect al clasei are un nou membru `*vptr` pointer către tabela dinamică `VTABLE` care conține poineri către implementările funcțiilor virtuale.

Despre metode virtuale:

- e recomandat să fie declarate public;
- metodele virtuale nu pot fi statice;
- pentru a putea folosi polimorfismul trebuie, e recomandat să fie apelate folosind pointeri sau referințe.
- Putem avea si destructori virtuali (ajută la distrugerea obiectelor asignate prin upcasting);

3 Clase abstracte

Atunci când declarăm o metodă virtuală nu suntem obligați să furnizăm mereu o implementare. Putem transforma metoda virtuală într-o *metodă virtuală pură* adăugând `= 0` după signatura

metodei.

```
class C {
public:
    virtual void foo() = 0;
};
```

Putem oferi implmenentări pentru metodele virtuale pure, iar singura metodă prin care acestea pot fi folosite sunt prin apelarea explicită în clase derivate.

```
#include<iostream>
using namespace std;
class B {
    public:
    virtual void foo() = 0;
};

void B::foo() {
    cout << "foo";
}


class D: public B {
public:
    void foo () {    // eliminarea acestei redefiniri
        B::foo();
    }
};
```

O clasă cu o metodă virtuală pură se numește *clasă abstractă*. O clasă abstractă nu poate fi instanțiată. Se pot declara doar pointeri și referințe către o astfel de clasă. Clasele care moștenesc o clasă abstractă trebuie să implementeze toate metodele virtuale pure.

Exerciții

1. (în continuarea exercițiului din laboratorul 6) Fiecare eveniment are un cost de administrare:

   - un eveniment individual costă 100 lei pe ora;
   - un eveniment recurent primeste un discount in funcție de recurența acestuia: 18% pt evenimente zilnice, 13% pt evenimente săptămânale, 7% pentru evenimente lunare 5% pentru evenimente anuale.
   - un eveniment de grup costă 30 lei pe oră de persoana.

2. Spuneți care dintre următoarele secvețe de cod compilează și care nu. În cazul secvențelor de cod care compileaza spuneți care este outputul programului. În cazul secvențelor care nu compilează sugerați o modifcare prin care secvența compilează și spuneți care este outputul secvenței modificate.

```
// 1
#include<iostream>
using namespace std;
class A {
    public:
    A(int x){cout<<"A"<<x;}
    ~A(){cout<<"~A";}
};

class B: public A {
public:
    B(int y=3){cout<<"B"<<4;}
};

```

```
15 class C: public B {
16     C ():B(10) {cout<<"C";}
17 }
18
19 int main () {
20     A *pa = new C();
21     delete pa;
22     return 0;
23 }
```

```cpp
// 2
#include<iostream>
using namespace std;

class A {
    int y;
public:
    A(int x = 2020) : y(x) {cout<<"A";}
    ~A(){cout<<"~A";}
    ostream& operator<<(ostream& out) const {
        out<<this->y; return out;
    }
};

int main () {
    A a(2), b;
    a << (b << cout);
    return 0;
}
```

```cpp
// 3
#include <iostream>
using namespace std;

class C {
    int * const p;
public:
    C(int x) : p(&x) {(*p)+=3;}
    void set (int x) {p = &x;}
    friend ostream& operator<<(ostream& o, C x) {
        o << *x.p; return o;
    }
};

int main () {
    cout << C(3);
    return 0;
}
```

```cpp
// 4
#include <iostream>
using namespace std;

#define ltlt <<

class B {
public:
    virtual void f() = 0;
};

void B::f(){cout ltlt "I'm so B";}

class C : public B {
};

int main () {
    B* pb = new C();
    return 0;
}
```

```cpp
// 5
#include <iostream>
using namespace std;

class A {
public:
    A () {cout<<"A";}
    virtual ~A() = 0;
};
A::~A(){cout<<"~A";}

class B : public A {
public:
    B(){cout<<"B";}
    ~B(){cout<<"~B";}
};

class C: public B {
public:
    C () {cout << "C";}
    ~C () {cout << "~C";}
};

int main () {
    C c;
    B &pb = c;
    return 0;
}
```