Лекція 5. Класти та об'єкти

Клас – базова одиниця інкапсуляції. Клас поєднує в собі дані (поля) та програмний код (методи) для їх обробки. Клас є шаблоном, за яким створюються об'єкти

Об'єкт – конкретний екземпляр класу

Члени класу - поля та методи класу

Члени класу можуть бути відкритими та закритими (за умовчанням)

Оголошення класу

```
class ім’я_класу{
//закриті поля та методи
public:
//відкриті поля та методи
}список_об'єктів;
```

Звернення до членів класу:

"точковий синтаксис":

ім'я_об'єкту.член_класу

методи вказуються з круглими дужками

Приклад 5.1. Створення і використання класу

```
#include<iostream>
using namespace std;
// Оголошення класу SimpleClass:
class SimpleClass{
public:
// Цілочисельне поле:
int number;
};
int main(){
// Створення об'єкту MyObj класу SimpleClass:
SimpleClass MyObj;
// Полю об'єкта присвоюється значення:
MyObj.number=5;
cout<<"Object field value is "<<MyObj.number<<"\n";
return 0;}
```

Приклад 5.2. Декілька об'єктів

ЗВЕРНІТЬ УВАГУ!!!

```
#include<iostream>
using namespace std;
class SimpleClass{
public:
int number;}MyObj1, MyObj2;
int main(){
MyObj1.number=5;
MyObj2.number=++MyObj1.number;
cout<<"Object field value is "<<MyObj2.number<<"\n";
return 0;}
```

Приклад 5.3. Декілька об'єктів

```
class SimpleClass{
public:
int m, n;

int summa(){
int k=n+m;
return k;}

void show(){
cout<<"m = "<<m<<endl;
cout<<"n = "<<n<<endl;}

void mult(int k){
n*=k;
m*=k;}
};
```

Приклад 5.3. Декілька об'єктів (продовження)

```
int main(){
SimpleClass MyObj1,MyObj2;

MyObj1.m=1;
MyObj1.n=2;
MyObj2.m=8;
MyObj2.n=9;

cout<<"Total vaue for MyObj1 is "<<MyObj1.summa()<<endl;
cout<<"Total value for MyObj2 is "<<MyObj2.summa()<<endl;

MyObj1.mult(3);
MyObj2.mult(2);
MyObj1.show();
MyObj2.show();

return 0;}
```

Приклад 5.4. Закриті і відкриті члени класу

```
class SimpleClass{
//Закриті члени класу:
int m;
int n;
public:
//Відкриті члени класу:
void show();
void setnm(int i,int j);
};
```

Приклад 5.4. Закриті і відкриті члени класу (продовження)

```
int main(){
SimpleClass obj;
obj.setnm(1,2);
obj.show();
return 0;}

//Опис методів класу:
void SimpleClass::show(){
cout<<"m = "<<m<<endl;
cout<<"n = "<<n<<endl;}

void SimpleClass::setnm(int i,int j){
m=i;
n=j;}
```

Статичні поля класу

Статичне поле класу - поле, спільне для всіх об'єктів цього класу. Оголошується за допомогою ключового слова **static**.

Приклад 5.5. Статичні поля класу

```
class SimpleClass{
public:
static int m;
int n;
void show();}obj1,obj2;
//Повторне оголошення статичного поля:
int SimpleClass::m;
```

Приклад 5.5. Статичні поля класу (продовження)

```
int main(){
SimpleClass::m=10;
obj1.n=1;
obj2.n=2;
obj1.show();
obj2.show();
obj1.m=100;
obj2.show();
return 0;}

void SimpleClass::show(){
cout<<"Static field m = "<<m<<endl;
cout<<"Nonstatic field n = "<<n<<endl;}
```

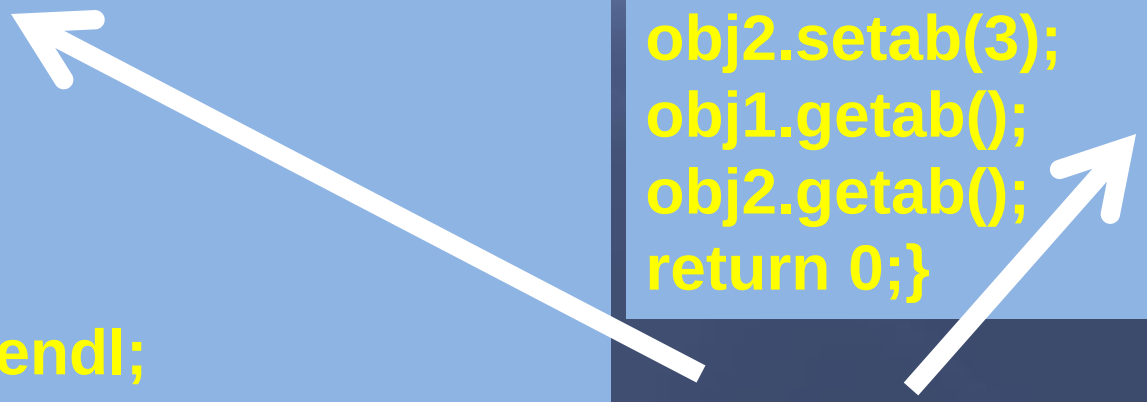
Перевантаження методів

Як і звичайні функції, методи класів можна перевантажувати. Перевантаження методу - створення декількох версій методу з одною і тою самою назвою, але різними прототипами

Приклад 5.6. Перевантаження методу

```
class MyClass{
int a,b;
public:
void setab(int i,int j){
a=i;
b=j;}
void setab(int i){
a=i;
b=i;}
void getab(){
cout<<"a = "<<a<<endl;
cout<<"b = "<<b<<endl;}
}obj1,obj2;
```

```
int main(){
obj1.setab(1,2);
obj2.setab(3);
obj1.getab();
obj2.getab();
return 0;}
```



ЗВЕРНІТЬ УВАГУ!!!

Передача об'єктів аргументами функцій

При передачі об'єкту аргументом функції в якості імені аргументу вказується ім'я відповідного класу

Приклад 5.7. Аргумент функції - об'єкт

```
class ClassA{
public:
double x,y;
}objA;
class ClassB{
public:
int a,b;
double f(ClassA obj){
return a*obj.x+b*obj.y;}
}objB;
```

```
void showB(ClassB obj){
cout<<"a = "<<obj.a<<endl;
cout<<"b = "<<obj.b<<endl;}
int main(){
objA.x=2.5;
objA.y=3.6;
objB.a=2;
objB.b=5;
cout<<"Value is "<<objB.f(objA)<<endl;
showB(objB);
return 0;}
```

Результат функції - об'єкт

Якщо функція в якості результату повертає об'єкт, ідентифікатором типу результату вказують ім'я відповідного класу

Приклад 5.8. Результат функції - об'єкт

```
class ClassA{
public:
double x;
double y;
}objA;

class ClassB{
public:
int a,b;
}objB;
```

Приклад 5.8. Результат функції - об'єкт (продовження)

```
ClassA sumAB(ClassA obj1,ClassB obj2){
ClassA tmp;
tmp.x=obj1.x+obj2.a;
tmp.y=obj1.y+obj2.b;
return tmp;}

int main(){
ClassA obj;
objA.x=2.5;
objA.y=3.6;
objB.a=2;
objB.b=5;
obj=sumAB(objA,objB);
cout<<"x = "<<obj.x<<endl;
cout<<"y = "<<obj.y<<endl;
return 0;}
```

ЗВЕРНІТЬ УВАГУ!!!

Вказівники на об'єкти

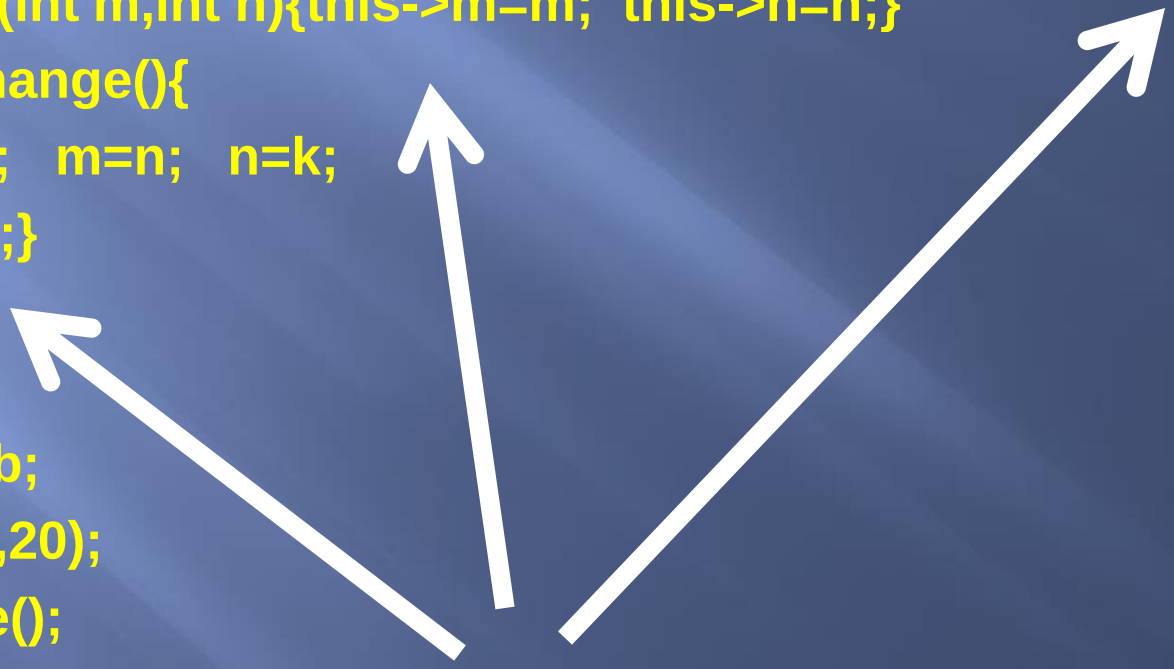
Змінна-вказівник може посилатись на об'єкт. Оператор -> використовують для посилання на член класу. Ключове слово this є вказівником на об'єкт, з якого викликається метод

Приклад 5.9. Вказівники на об'єкти

```
class MyClass{
public:
int n;
void show(){cout<<"n = "<<n<<endl;}
};
int main(){MyClass a,b;
//Вказівники на об'єкт:
MyClass *p,*q;
//Значення вказівників:
p=&a;    q=&b;
//Доступ до полів і методів через вказівник:
p->n=10;   (*q).n=20;   p->show();   (*q).show();
return 0;}
```

Приклад 5.10. Використання вказівника this

```
class MyClass{
public:
int n,m;
void show(){cout<<"m = "<<this->m<<endl; cout<<"n = "<<this->n<<endl;}
void setmn(int m,int n){this->m=m;  this->n=n;}
MyClass change(){
int k;   k=m;   m=n;   n=k;
return *this;}
};
int main(){
MyClass a,b;
a.setmn(10,20);
b=a.change();
a.show();
b.show();
return 0;}
```



ЗВЕРНІТЬ УВАГУ!!!

Масиви об'єктів

При створенні масиву об'єктів в якості типу елементів вказують ім'я відповідного класу

Приклад 5.11. Масиви об'єктів

```
class MyClass{
public:
double x;
void show(){cout<<"x = "<<x<<endl;}
};
int main(){
const n=5;
int i;
MyClass objs[n];
for(i=0;i<n;i++){
objs[i].x=2*i+1;
cout<<i+1<<": ";
objs[i].show();}
return 0;}
```

ЗВЕРНІТЬ УВАГУ!!!

Дружні функції

В загальному випадку зовнішня до класу функція доступу до закритих членів класу не має. Щоб функція мала доступ до закритих членів, її необхідно оголосити як дружню до класу. Для цього прототип функції, разом з ключовим словом friend вказується при описі класу

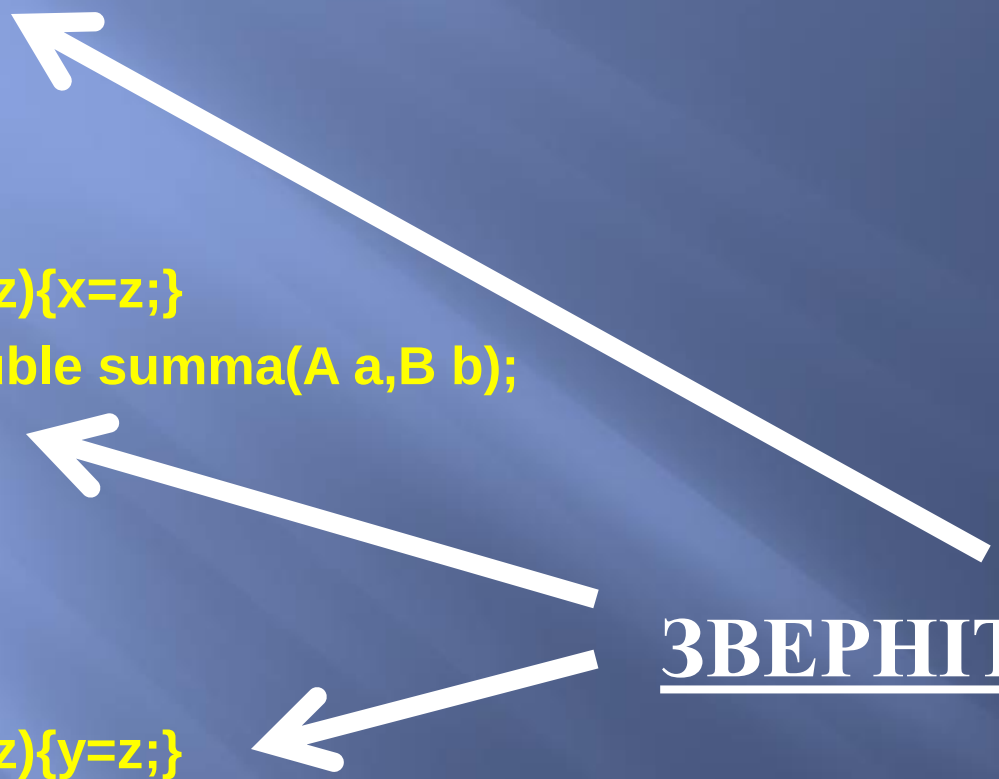
Приклад 5.12. Дружні функції

```
class MyClass{
double x;
public:
MyClass(double z){x=z;}
//Дружня функція:
friend void show(MyClass obj);
};
void show(MyClass obj){cout<<"x = "<<obj.x<<endl;}
int main(){
MyClass a(10);
show(a);
return 0;}
```

ЗВЕРНІТЬ УВАГУ!!!

Приклад 5.13. Дружні функції - 2

```
class B;
class A{
double x;
public:
A(double z){x=z;}
friend double summa(A a,B b);
}a(3.5);
class B{
double y;
public:
B(double z){y=z;}
friend double summa(A a,B b);
}b(2.3);
double summa(A a,B b){
return a.x+b.y;}
int main(){cout<<"Total is "<<summa(a,b)<<endl;
return 0;}
```



ЗВЕРНІТЬ УВАГУ!!!

Конструктори і деструктори

Конструктор - метод, що викликається автоматично при створенні об'єкту

Деструктор - метод, що викликається автоматично при видаленні об'єкту з пам'яті

Правила оголошення конструктора

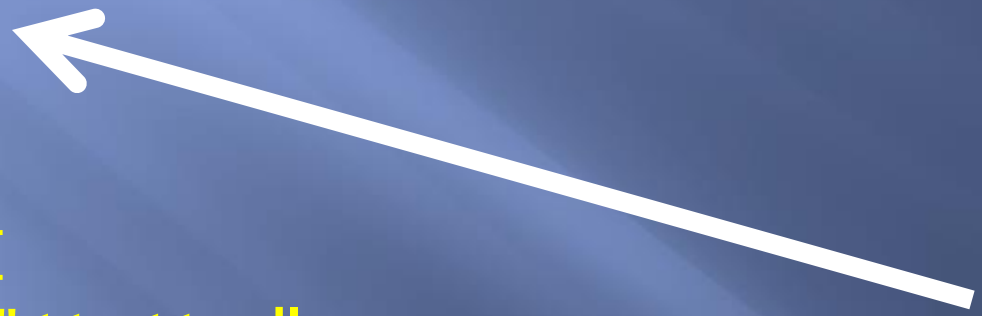
1. Ім'я конструктора співпадає з назвою класу
2. Конструктор не повертає аргумент і для нього не вказується ідентифікатор типу результату
3. Конструктор може мати аргументи і його можна перевантажувати

Правила оголошення деструктора

1. Ім'я деструктора починається з символу ~ (тильда) і співпадає з назвою класу
2. Деструктор не повертає аргумент і для нього не вказується ідентифікатор типу результату
3. Деструктор не має аргументів і його не можна перевантажувати

Приклад 5.14. Оголошення конструктора

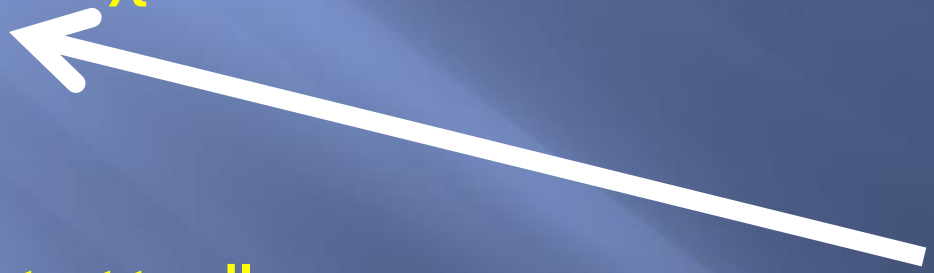
```
class MyClass{
public:
int m,n;
//Конструктор класу:
MyClass(){
m=0;
n=0;}
void show(){
cout<<"m = "<<m<<endl;
cout<<"n = "<<n<<endl;}
};
int main(){
//При створенні класу поля отримують значення:
MyClass obj;
//Відображення полів об'єкту:
obj.show();
return 0;}
```



ЗВЕРНІТЬ УВАГУ!!!

Приклад 5.15. Конструктор з аргументами

```
class MyClass{
public:
int m,n;
//Конструктор класу з аргументами:
MyClass(int a,int b){
m=a;
n=b;}
void show(){
cout<<"m = "<<m<<endl;
cout<<"n = "<<n<<endl;}
};
int main(){
//При створенні об'єкту вказуються значення полів:
MyClass obj(1,2);
//Відображення значень полів об'єкта:
obj.show();
return 0;}
```



ЗВЕРНІТЬ УВАГУ!!!

Приклад 5.16. Перевантаження конструктора

```
class MyClass{
public:
int m,n;
MyClass(){m=0;  n=0;}
MyClass(int a){m=a;  n=a;}
MyClass(int a,int b){m=a;  n=b;}
void show(){
cout<<"m = "<<m<<endl;
cout<<"n = "<<n<<endl;}
};
int main(){
MyClass obj1;
MyClass obj2(1);
MyClass obj3(2,3);
obj1.show();     obj2.show();    obj3.show();
return 0;}
```

ЗВЕРНІТЬ УВАГУ!!!

Приклад 5.17. Використання деструктора

```
class MyClass{
public:
int m,n;
//Конструктор класу:
MyClass(){
m=0;
n=0;
cout<<"Object has been created"<<endl;}
//Деструктор класу:
~MyClass(){
cout<<"Object has been deleted"<<endl;}
};
int main(){
MyClass obj;
return 0;}
```

ЗВЕРНІТЬ УВАГУ!!!

Резюме

1. Клас є базовою одиницею інкапсуляції, містить поля (дані) та методи (код для обробки даних). Екземпляром класу є об'єкт.

2. Поля і методи класу можуть бути закритими і відкритими.

3. Статичне поле - це поле, спільне для всіх об'єктів даного класу.

4. Щоб зовнішня функція мала доступ до закритих полів, її необхідно оголосити як дружню.

5. Об'єкти можна передавати аргументами функцій, повертати в якості результату, створювати масиви об'єктів, створювати вказівники на об'єкти.

6. Методи класів можна перевантажувати.

7. Конструктор - метод, що запускається при створенні об'єкту. Конструктор можна перевантажувати.

8. Деструктор - метод, що запускається при знищенні об'єкту. Деструктор не можна перевантажувати.