Инкапсуляция в C++

Понятие инкапсуляции

- **Инкапсуляция** - есть объединение в едином объекте данных и кодов, оперирующих с этими данными. В терминологии ООП данные называются *членами данных (data members)* объекта, а коды - объектными *методами или функциями-членами (methods, member functions)*.
- Инкапсуляция позволяет в максимальной степени изолировать объект от внешнего окружения. Она существенно повышает надежность разрабатываемых программ, т.к. локализованные в объекте функции обмениваются с программой сравнительно небольшими объемами данных, причем количество и тип этих данных обычно тщательно контролируются. В результате замена или модификация функций и данных, инкапсулированных в объект, как правило, не влечет за собой плохо прослеживаемых последствий для программы в целом (в целях повышения защищенности программ в ООП почти не используются глобальные переменные).
- Другим немаловажным следствием инкапсуляции является легкость обмена объектами, переноса их из одной программы в другую. Простота и доступность принципа инкапсуляции ООП стимулирует программистов к расширению Библиотеки Визуальных Компонент (VCL), входящей в состав C++Builder.

Понятие абстракции данных в C++

- **Абстра́кция** в объектно-ориентированном программировании — это придание объекту характеристик, которые чётко определяют его концептуальные границы, отличая от всех других объектов. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.

```
#include <iostream>
class MyClass
{ public: MyClass() = default;
~MyClass() = default;
inline void setX(int n) { x = n ; }
 inline void setY(int n) { y = n ; }
 inline int getX() { return x; }
 inline int getY() { return y; }
protected: int x;
private: int y; };
class Derived : public MyClass
{ public:
Derived() = default;
~Derived() = default;
inline void showX() { std::cout << x << "\n"; } ;
inline void showY() { std::cout << y << "\n"; } };
int main()
{ int a;
Derived t;
std :: cin >> a;
t.setX(a);
t.showX();
return 0; }
```