Пример

```
#include <iostream>

class C {
private:
    class Inner {
    public:
        int x = 1;
    private:
        int y = 2;
    };
public:
    Inner f() {
        return Inner();
    }
};
int main() {
    C c;
    std::cout << c.f().x; // CE or 1?
}
```

Хороший пример на понимание модификаторов доступа.

Будет 1, так как приватность запрещает именно называть **Inner** и обращатсья к нему, но если мы уже имеем объект, то можем с ним работать не обращая внимания на приватность класса **C**, можно так же сохранить его через **auto**.

Пример

```
#include <iostream>

class C {
private:
    void f(int) {
        std::cout << 1;
    }
public:
    void f(float) {
        std::cout << 2;
    }
};

int main() {
    C c;
    c.f(1); // CE or 2?
    c.f(1.0); // CE or 2?
}
```

В первом случае будет **CE**, так как это помогает проектированию классов, по задумке автора - неправильно вызывать `f(int)`, но `f(float)` можно. Фактически мы показываем с какими аргументами можно вызываться, а с какими нельзя.

Во втором случае тоже будет **CE**, так как мы вызываемся от `double`, нет разницы сконвертировать в **int или float** - это равнозначные конверсии, неоднозначная перегрузка - CE.

Функции и классы-друзья

Функции и классы-друзья - такие функции или классы, которые хоть и не являются членами данного класса, тем не менее им разрешен доступ к приватной части данного класса, такое отношение **не симметрично и не транзитивно**.

```
#include <iostream>

class C {
private:
    int x{5};
public:
    void f(int y) {
        std::cout << x + y;
    }

    friend void g(C, int);
    friend class CC;
};

void g(C c, int y) {
    std::cout << c.x + y + 1;
}

int main() {
    C c;
    std::cout << (int&)c;
}
```

`friend` - специальное ключевое слово, можно объявлять где угодно в классе, независимо от **private или public** (можно также и определить в классе, в данном случае было бы **error: redefinition**, соответсвенно можно определять отдельно в классе, но не рекомендуется).

Принцип работы: слово, которое говорит компилятору запомнить эту функцию, и когда будет определение этой функции, то ей разрешено обращение к приватным полям (анаголично и **классы**).

`friend` функции можно объявлять только после определения класса, иначе будут неизвестны поля класса и это не имеет смысла.

Если хотим подружить 2 класса, то у одного из них надо заранее определить поля, методы, using и тд.

`friend` не стоит использовать часто, скорее исключение из правил.

Конструкторы и деструкторы (ctor / dtor)