Синглтон

Назначение - создание строго одного объекта в рамках одного процесса, обеспечение доступа к его экземпляру в любом месте программы (улучшенная глобальная переменная).

Диаграмма классов на draw.io:

https://drive.google.com/file/d/1r1gpw8ih27fRl8dacYUcnwysOPB9C24Q/view?usp=sharing

```
class Singleton{
    static Singleton *p_instance;
    Singleton(){};
    Singleton(Singleton *);
    Singleton &operator=(Singleton &);
    public:
    static Singleton *getInstance(){
        if (p_instance){
            p_instance = new Singleton();
        }
        return p_instance;
    }
}

Singleton *Singleton::p_instance = 0;
int main(){
    Singleton *p;
    p = Singleton::get_Instance();
    p->connect();
}
```

Используются при:

- реализации курсора базы данных
- реализации логгирования
- организации различных синхронизирующих очередей

Есть модификация. называемая Синглтон Мейерса.

```
class Singleton{
    Singleton(){};
    Singleton(Singleton *);
    Sinbleton &operator=(Singleton &);
    public:
    static Singleton getInstance(){
        static Singleton instance;
        return instance;
    }
}
```

Недостаток - невозможность в ходе работы программы удалить объект. Для избавления от этого недостатка можно использовать парные конструкторы/деструкторы.

Полиморфные синглтоны

```
class Singleton;
class SingletonDestroyer{
    public:
    void init(Singleton *p);
    ~SingletonDestroyer();
    private:
    static Singleton *p;
}

class Singleton{
    friend class SingletonDestroyer;
    protected:
    Singleton(){};
    Singleton(Singleton *);
    Sinbleton &operator=(Singleton &);
    public:
    static Singleton getInstance(){
        if (!instance){
            instance = new Singleton();
            destroy.init(instance);
        }
        return instance;
    }
    private:
    static Singleton *instance;
    Static SingletonDestroyer destroy;
}
```

Паттерн: прототип

https://drive.google.com/file/d/1ETRnDP8Frm2njFVKunfid68_zMt82ACK/view?usp=sharing

```
enum TypeUnit{
    Warrior_Id = 0;
    Archer_Id = 1;
}

typedef map <TypeUnit, Unit*> Reg;

Reg &getKey(){
    static Reg r;
    return r;
}

```

```
class Unit{
    public:
    virtual clone() = 0;
    static Unit *createUnit(
        TypeUnit d){
        Reg&r = getReg()
        }
    if (r.find(ID) == r.end(){
        return r[d] =>clone();
        return 0;
    )
}
}
```