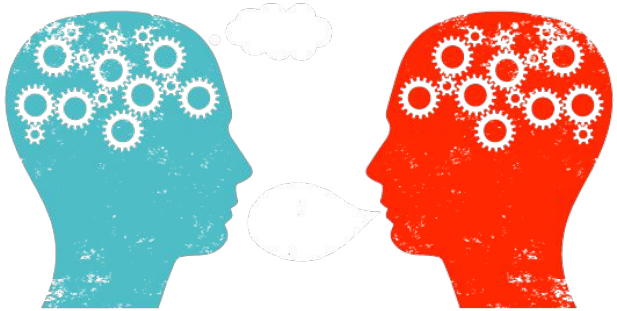
OOP - Organization & Methods

Organization

- Two file types are needed: .cpp & .h

- .cpp file → Normal C++ file
  - Includes #include statements, main(), etc...
  - Will be the “engine” of your code

Organization



- Two file types are needed: .cpp & .h

- .h file → Header files
  - ONE HEADER FILE PER CLASS
  - You will need to make your own .h files (in addition to the ones created by C++)
  - Are added to your .cpp file by **#include <filename.h>**

Organization → .h files

- The .h file will be a *declaration* of user-defined classes
  - NO specific code for methods
  - NO values for variables
  - Simply declares that methods & variables exist

```
/*File named movie.h*/

class movie {
    public:
        string getActors();
        string getActress();
        void setRelease();
        double getRevenue();
    private:
        string actor;
        string name;
        int release;
        double revenue;
};
```

Organization → .cpp files

- Includes the *definition* of the methods declared in a .h file
- This definition goes directly above main

C++ Program Structure

```
#include statements
Method definitions
main()
```

Organization → Method definitions

- Method definition format
  - **Class_name + “::” + method_name (variables)**

```
string Movie::getActors(void) {
   /*Code that defines the getActors() method*/
}
```

```
#include statements
Method definitions
main()
```

OOP - Methods

- Four types of methods:
  - Accessors
  - Mutators
  - Facilitators
  - Constructors

OOP- Accessors

- **Accessor** methods are used to access the value of private variables

```
>> int movie::getRevenue () {
        return revenue;
}
```

OOP - Mutators

- Mutators **change** the value of a private variable

```
>> void movie::changeRevenue(int num){
        revenue = num;
}
```

OOP - Facilitators

- Facilitators allow some “service” to happen

```
>> void movie::multiply_Revenue(int num){
        revenue = revenue * num;
}
```

OOP - Constructors

- Allow for an object to be created with variables already created
  - Has the **same name** as the class!

```
>> class movie {
       public:
            movie (string init_name, double init_revenue);
    };
```

OOP - Constructors

- Allow for an object to be created with variables already created

```
>> void movie::movie(string init_name, double init_revenue) {
        name = init_name;
        revenue = init_revenue;
```

OOP - Defining Objects

- Objects are instances of classes
- Can be instantiated in main()
  - Or in any other method...

- Defined similarly to structs, but with a constructor behind the scenes

```
>> movie RogueOne("Rogue One", 2160000000);
```

OOP - File Setup

- C++ uses .h and .cpp files during compilation
  - .h files hold **class definitions**
  - .cpp files hold **main() and class functions**

.h File

```
using namespace std;
class movie {
        public: /*all public code*/
        private: /*all private
code*/
};
```

.cpp File

```
void movie::getRevenue() {
        /*Function definition*/
}

int main() { /*etc…*/ }
```

OOP - File Setup

- .cpp files MUST #include all .h files used within a file
  - #include <movie.h>

.h File

```
using namespace std;
class movie {
        public: /*all public code*/
        private: /*all private code*/
};
```

.cpp File

```
#include <iostream>
using namespace std;

Constructors & other methods

int main() { /*etc…*/ }
```

Coding(?) Challenge

- Create a class definition of a song in a .h file, and create the outline of the definitions in a .cpp file (DO NOT FULLY CODE YOUR .cpp FILE)
  - Include any variables/methods necessary to describe a song
    Include a constructor and accessors (for every variable) at minimum