**ВИСШЕ ВОЕННОМОРСКО УЧИЛИЩЕ “Н. Й. ВАПЦАРОВ”**

***9002 Варна, ул. “В. Друмев” 73, тел. 052/632-015, факс 052/303-163***

**═══════════════════════**

***“FILII MARIS SUMUS”***

**ФАКУЛТЕТ “ИНЖЕНЕРЕН” - КАТЕДРА “ИНФОРМАЦИОННИ ТЕХНОЛОГИИ”**

**КУРСОВ ПРОЕКТ**

**по дисциплина “Курсов проект по дисциплина от 2-ри или 3-ти семестър”**

**ТЕМА: CLI игра - симулация на изследователска дейност на отдалечена планета**

**Студент: Петър Димитров Попов**

**Специалност: Информационни и комуникационни технологии**

**Фак. № 111 - 7419**

**гр. Варна**

**2020 г.**

Програмен език: **C++**

Source код: <https://github.com/p-dim-popov/SpaceExploration-CLI_Game>

**Увод**

Идеята ми за този проект е да се опитам да направя работещо приложение, което е максимално разделено на модули. Всеки един от които може да бъде заменен с друг използвайки наследяване на интерфейсните класове, които изграждат приложението. Всеки клас на това приложение ще използва интерфейси като това ще позволи пълното използване на полиморфизма, абстракцията и наследяването, които езикът C++ ни предоставя. Където е възможно ще бъдат спазени и S.O.L.I.D. принципите на обектно ориентираното програмиране.

1. **Ще разгледам гореспоменатите S.O.L.I.D. принципи:**

* Single responsibility principle (принцип на единичната отговорност) - в ООП Single Responsibility Principle гласи, че всеки модул и/или клас трябва да има отговорност върху точно една част от функционалността, която софтуерът предлага
* Open/closed principle (принцип на отворената затвореност) - в ООП open/closed principle гласи, че всички софтуерни части (класове, модули, функции и т.н.) трябва да бъдат отворени за разширяване, но затворени за модификации. Това се осъществява с помощта на гореспоменатото използване на полиморфизъм, абстракция и наследяване. Декларират се интерфейси, които гарантират на програмата, че всеки техен наследник ще реализира декларираните методи и прочие.
* Liskov substitution principle - (принцип за субституцията на Лисков) - този принцип е един от най-трудните за първоначално осмисляне. В ООП принципът за субституция на Лисков гласи, че ако Y е подтип на X, то всички обекти от типа X могат да бъдат заменени с обекти от типа Y без да променят верността на дадената програма. Това може да се сложи в математически вид по следния начин:
  + *- Нека f(x) е свойство, доказуемо за обекти x от типа X, тогава f(y) е вярно за обекти y от тип Y, където Y е подтип на X.*
* Interface segregation principle (принцип за сегрегация на интерфейсите) - този принцип е лесен за разбиране. В ООП interface segregation principle гласи, че нито един клиент не трябва да е принуден да зависи от методи, които не ползва. По-точно - да се избягва добавянето на нова функционалност на съществуващ интерфейс. При такава необходимост се създава нов интерфейс и се налага наследяване на този интерфейс от класовете наследници. В С++, за разлика от С#, където не се позволява множествено наследяване (тази необходимост е премахната, тъй като се осъществява множествено наследяване на интерфейсни класове), тук един клас може да наследява много други класове.
* Dependency inversion principle (принцип за обратната зависимост) - в ООП идеята на dependency inversion principle е да се осъществи това “разделяне” на софтуерните модули. Този принцип гласи следните неща:
  + Модули от високо ниво не трябва да зависят от модули на ниско ниво. И двата вида трябва да зависят на абстракции.
  + Абстракциите не трябва да зависят на детайли. Детайлите трябва да зависят на абстракции

Осъществяването на този принцип се прави чрез така нареченото “инжектиране на зависимости” като всички зависимости на даден клас се подават като параметри към неговият конструктор.

Реализацията на тези 5 принципа наречени S.O.L.I.D. (в превод “стабилност”) позволяват на едно приложение да бъде преизползваемо, лесно поддържано, скалируемо и лесно тестващо се.

1. **Сюжет**

Главният герой сте вие и целта е да съберете екип от космонавти, за да се изпратят в космоса. Добавят се в космически кораб, добавят се планети за изследване и предмети на всяка планета. Задава се планета за изследване и се започва със събиране на предмети. Някои астронавти могат да умрат, други могат да оцелеят. Космосът е опасно място. Когато главният герой реши, може да пенсионира астронавт, който е рискувал живота си достатъчно в името на родината.

Започва се със събиране на екип от астронавти, спадащи към следните категории:

* Биолог - начални единици кислород: 70
* Геодезист - начални единици кислород: 65
* Метеоролог - начални единици кислород: 90

Командата, чрез която се добавят астронавти, е:

|  |
| --- |
| ДобавиАстронавт <длъжност/категория> <име на астронавт> |

Всеки астронавт притежава по една раница, която побира до 3 предмета.

Когато астронавт открие предмет, неговото количество кислород намалява с 10 единици. Благодарение на тайно откритие на биолозите, техният кислород намалява само с 2 единици при намерен предмет.

Въвеждат се планети и техните предмети, които астронавтите ще трябва да намерят и съберат по време на експедицията.

Командата, чрез която се добавят планети е:

|  |
| --- |
| ДобавиПланета <име на планета> [предмет\_1,предмет\_2, … предмет\_X] |

След това се посочва планета за изследване чрез командата:

|  |
| --- |
| ИзследвайПланета <име на планета> |

Избират се астронавти, годни за експедиция, такива които имат над 20 единици кислород и имат място в раницата си. Когато експедицията на дадена планета започне, астронавтите се приземяват и започват да търсят предмети един по един. Редът се определя случайно. При намиране на елемент, кислородът на дадения астронавт намалява, като има 33% вероятност, поради затруднения при взимането на предмета от планетата, астронавтът да изхаби двойно повече кислородни единици. Действието продължава докато или на планетата има неоткрити предмети, или кислородът на всички астронавти се изчерпа, или нямат повече място в раниците си. След това оцелелите астронавти се прибират.

Поради опасностите на открития космос има 50% вероятност всички астронавти, изпратени на дадена експедиция, да загинат или изчезнат.

Всеки един астронавт може да бъде пенсиониран и изпратен обратно в дома си по всяко време с командата:

|  |
| --- |
| ПенсионирайАстронавт <име на астронавт> |

Редът на действията, оказан дотук, е само примерен. Играта има механизми за справяне с грешен ред на действията чрез известяване на потребителя или игнориране на грешни команди.

**По-подробно описание на командите**: Играта се състои от няколко команди, чрез които се контролира редът на действията. Работи на принципа на команден интерпретатор. Командите са:

*ЛЕГЕНДА: [value] - описва опционален аргумент, <value> - описва задължителен аргумент:*

* ДобавиАстронавт <име на астронавт> <длъжност/категория> //добавя астронавт към съществуващата база данни от абстрактният клас Astronaut
* ДобавиПланета <име на планета> [предмет\_1,предмет\_2, … предмет\_X] //Добавя планета към данните на обекта PlanetRepository. Записването на предметите става чрез разделител “,” БЕЗ разстояние
* ПенсионирайАстронавт <име на астронавт> //премахва астронавт от гореспоменатата БД
* ИзследвайПланета <име на планета> //инициализира избор на астронавти за експедиция до посочената планета
* Отчет //показва текущото състояние на астронавтите (Име, Количество кислород, Предмети)
* Край //приключва програмата

//Всяка грешно въведена команда се игнорира и програмата очаква нов вход на данни

Програмата работи и в среда на Microsoft Windows, но въвеждането на командите се извършва на английски език.

* AddAstronaut <type> <name>
* AddPlanet <name> [item1,item2, … itemN]
* RetireAstronaut <name>
* ExplorePlanet <name>
* Report
* End

**Изложение**

Ще разгледам само най-важните части на програмата.

1. **Функцията main**

|  |
| --- |
| 01 #include"engine.hpp"  02 #include **<**iostream**>**  03  04 **using** **namespace** std**;**  05  06 int main**()** **{**  07 #ifdef \_WIN32  08 system**(**"chcp 65001"**);**  09 #endif  10 srand**(**time**(nullptr));**  11 #ifndef \_\_EMSCRIPTEN\_\_  12 AstronautRepository astronauts**;**  13 PlanetRepository planets**;**  14 Mission mission**;**  15 Engine eng**(**cin**,** cout**,** astronauts**,** planets**,** mission**);**  16 eng**.**run**();**  17 #endif  18 **return** 0**;**  19 **}** |

Програмата започва с няколко проверки за средата, в която се изпълнява. На ред 7 се проверява за наличие на дефиниция \_WIN32, която се дефинира като 1, когато програмата се компилира в среда на MS Windows. Ако \_WIN32 е дефинирана, се преминава към задаване на UTF-8 кодиране за Windows’ CMD.

На 10-ти ред се генерира “случаен” сийд според времето на компилация, който е нужен за пресмятането на случайни събития по-късно в играта.

Програмата проверява за наличието на дефиниция \_\_EMSCRIPTEN\_\_. Тя бива дефинирана, когато се използва компилатор Emscripten. Това е компилатор, който осъществява компилация на C++ код към JavaScript, което позволява по-нататъшно изпълнение на тази игра в среда на браузър, което я прави максимално мултиплатформена.

Ако \_\_EMSCRIPTEN\_\_ не е дефинирана, означава, че играта ще се компилира като изпълним файл и е нужно инстанцирането на няколко обекта от следните класове:

* AstronautRepository astronauts - хранилище за астронавти, в което ще бъдат съхранявани всички въведени астронавти
* PlanetRepository planets - хранилище за планети, в което ще бъдат съхранявани всички въведени планети
* Mission mission - мисия, която ще се играе
* Engine eng - интерпретатор на въвежданите команди. Това е един от интересните класове в тази програма и той ще бъде разгледан след тази точка (в 2. Класът Engine). Към него се подават конзола за въвеждане cin, конзола за извеждане cout, хранилището на астронавти - astronauts, хранилището за планети - planets и мисията, която ще се играе - mission;

Те биват вмъкнати в source кода посредством директивата #include “engine.hpp”, която вмъква хедъра engine.hpp. В него са използвани останалите части на това приложение

На 16-ти ред програмата започва своето изпълнение и вече очаква вход от потребителя.

При успешно изпълнение главната функция завършва с код 0 (успех).

1. **Класът Engine**

За да разгледам успешно Engine, ще бъде най-удачно да започна от интерфейса, който той наследява - IEngine.

|  |
| --- |
| 01 #ifndef IENGINE\_H  02 #define IENGINE\_H  03 class IEngine  04 **{**  05 public**:**  06 virtual void run**()** **=** 0**;**  07  08 virtual **~**IEngine**()** **=** **default;**  09 **};**  10 #endif // IENGINE\_H |

IEngine е интерфейс (чист абстрактен клас), чиято цел е да бъде наследяван от класове, за да служи като “договор”, че класът наследник ще имплементира функциите на интерфейса. Така се позволява по-нататъшно използване на интерфейса вместо наследника за осъществяването на S.O.L.I.D. принципите чрез полиморфизма.

IEngine декларира функцията virtual void run**()** **=** 0**;** и с това задължава всеки негов наследник да имплементира тази функция.

|  |
| --- |
| 01 #ifndef ENGINE\_H  02 #define ENGINE\_H  03 #include"iEngine.hpp"  04 #include**<**iostream**>**  05 #include**<**istream**>**  06 #include**<**ostream**>**  07 #include**<**fstream**>**  08 #include"controller.hpp"  09  10 class Engine **:** public IEngine  11 **{**  12 std**::**istream **\***in**;**  13 std**::**ostream **\***out**;**  14 IRepository**<**Astronaut**>\*** astronauts**;**  15 IRepository**<**Planet**>\*** planets**;**  16 IMission**\*** mission**;**  17  18 public**:**  19 Engine**(**std**::**istream**&** in**,** std**::**ostream**&** out**,** IRepository**<**Astronaut**>&** astronauts**,** IRepository**<**Planet**>&** planets**,** IMission**&** mission**);**  20 void run**()** override**;**  21  22 **~**Engine**()** override**;**  23 **};**  24 #endif // ENGINE\_H |

Engine наследява IEngine, с което съгласява и имплементира void run**()** и я означава override, за да означи, че е наследена функция.

Сега ще разгледам самата имплементация на run от Engine

|  |
| --- |
| 01 #include "engine.hpp"  02  03 Engine**::**Engine**(**std**::**istream **&**in**,** std**::**ostream **&**out**,** IRepository**<**Astronaut**>** **&**astronauts**,** IRepository**<**Planet**>** **&**planets**,**  04 IMission **&**mission**)** **{**  05 **this->**in **=** **&**in**;**  06 **this->**out **=** **&**out**;**  07 **this->**astronauts **=** **&**astronauts**;**  08 **this->**planets **=** **&**planets**;**  09 **this->**mission **=** **&**mission**;**  10 **}**  11  12 void Engine**::**run**()** **{**  13 Controller c**(\*this->**astronauts**,** **\*this->**planets**,** **\*this->**mission**);**  14  15 **while** **(true)**  16 **{**  17 std**::**string input**;**  18 std**::**getline**(**std**::**cin**,** input**);**  19 **if(**input **==** **(**"Край"**)** **||** input **==** **(**"End"**))**  20 **break;**  21 std**::**vector**<**std**::**string**>** args**;**  22 std**::**stringstream split\_stream**(**input**);**  23 std**::**string token**;**  24 **while** **(**std**::**getline**(**split\_stream**,** token**,** ' '**))** **{**  25 args**.**push\_back**(**token**);**  26 **}**  27 auto command **=** args**[**0**];**  28 **try** **{**  29 **if(**command **==** **(**"ДобавиАстронавт"**)** **||** command **==** **(**"AddAstronaut"**))**  30 **{**  31 **\*this->**out **<<** c**.**addAstronaut**(**args**[**1**],** args**[**2**]);**  32 **}**  33 **else** **if(**command **==** **(**"ДобавиПланета"**)** **||** command **==** **(**"AddPlanet"**))**  34 **{**  35 std**::**vector**<**std**::**string**>** items**;**  36 **if** **(**args**.**size**()** **==** 3**)**  37 **{**  38 std**::**stringstream split\_stream\_items**(**args**[**2**]);**  39 std**::**string item\_token**;**  40 **while** **(**std**::**getline**(**split\_stream\_items**,** item\_token**,** ','**))** **{**  41 items**.**push\_back**(**item\_token**);**  42 **}**  43 **}**  44 **\*this->**out **<<** c**.**addPlanet**(**args**[**1**],** items**);**  45 **}**  46 **else** **if(**command **==** **(**"ПенсионирайАстронавт"**)** **||** command **==** **(**"RetireAstronaut"**))**  47 **{**  48 **\*this->**out **<<** c**.**retireAstronaut**(**args**[**1**]);**  49 **}**  50 **else** **if(**command **==** **(**"ИзследвайПланета"**)** **||** command **==** **(**"ExplorePlanet"**))**  51 **{**  52 **\*this->**out **<<** c**.**explorePlanet**(**args**[**1**]);**  53 **}**  54 **else** **if(**command **==** **(**"Отчет"**)** **||** command **==** **(**"Report"**))**  55 **{**  56 **\*this->**out **<<** c**.**report**();**  57 **}**  58 **else** **\*this->**out **<<** "Грешна команда"**;**  59 **}**  60 **catch** **(**std**::**exception**&** e**)**  61 **{**  62 **\*this->**out **<<** e**.**what**();**  63 **}**  64 **}**  65 **}**  66  67 Engine**::~**Engine**()** **=** **default;** |

Функцията run започва с инстанциране на класът Controller (ще бъде разгледан в 3. Класът Controller), който приема подадените през конструктора на Engine, хранилища на астронавти и планети, а също така и мисията, която ще се играе. Така се осъществява dependency injection и се суплементират необходимите инстанции на Controller класа.

На ред 15 се преминава към while цикъл, по времето на който, ще се осъществи цялата игра.

Декларира се променлива за вход и се изчита входния ред от потребителя. Ако е въведено “Край” или “End” цикълът спира и функцията приключва.

От ред 21-ви до ред 27-ми се осществява разделянето на входните данни по разделител ‘ ’. Започва се с декларация на контейнер vector от низове args, след това входния низ се подава на декларираният низов поток на 22-ри ред split\_stream и след това се декларира временна променлива за запазване на всеки фрагмент от входния низ. С while цикъл, изпълняващ се докато съществува гореспоменатия разделител, се подава всеки един фрагмент на контейнера args.

Декларира се името на командата с променливата command, като се взима първия фрагмент от контейнера.

Всички функции от Controller-a “хвърлят” грешка когато нещо не сработи както трябва и затова на ред 28-ми започва така нареченият “try/catch блок”, където се изпълнява код, който се очаква да “хвърли” грешка.

Въведената команда се сравнява с възможните (ДобавиАстронавт, ДобавиПланета, ПенсионирайАстронавт, ИзследвайПланета, Отчет или техните алтернативи AddAstronaut, AddPlanet, RetireAstronaut, ExplorePlanet, Report) и се преминава към изпълнение на съответната. Всяка команда извиква функция на Controller-a, така че ще бъдат разгледани подробно в “3. Класът Controller”. Ако въведената команда не е разпозната, Engine-a извежда съобщение “Грешна команда” и се връща към повторно въвеждане.

Ако по време на изпълнението някоя функция върне грешка, то тя се “хваща” и се изписва на конзолата и цикълът продължава своята работа.

1. **Класът Controller**

Controller, също както Engine, наследява интерфейс, затова първо ще разгледам IController

|  |
| --- |
| 01 #ifndef ICONTROLLER\_H  02 #define ICONTROLLER\_H  03 #include**<**string**>**  04 #include**<**vector**>**  05 class IController  06 **{**  07 public**:**  08 virtual std**::**string addAstronaut**(**const std**::**string**&** type**,**const std**::**string**&** astronautName**)** **=** 0**;**  09 virtual std**::**string addPlanet**(**const std**::**string **&** planetName**,** std**::**vector**<**std**::**string**>&** items**)** **=** 0**;**  10 virtual std**::**string retireAstronaut**(**const std**::**string **&**astronautName**)** **=** 0**;**  11 virtual std**::**string explorePlanet**(**const std**::**string **&**planetName**)** **=** 0**;**  12 virtual std**::**string report**()** **=** 0**;**  13  14 virtual **~**IController**()** **=** **default;**  15 **};**  16 #endif // ICONTROLLER\_H |

IController всъщност декларира функциите, представляващи командите на играта:

* virtual std**::**string addAstronaut**(**const std**::**string**&** type**,**const std**::**string**&** astronautName**)** - ДобавиАстронавт
* virtual std**::**string addPlanet**(**const std**::**string **&** planetName**,** std**::**vector**<**std**::**string**>&** items**)** - ДобавиПланета
* virtual std**::**string retireAstronaut**(**const std**::**string **&**astronautName**)** - ПенсионирайАстронавт
* virtual std**::**string explorePlanet**(**const std**::**string **&**planetName**)** - ИзследвайПланета
* virtual std**::**string report**()** - Отчет

|  |
| --- |
| 22 class Controller **:** public IController **{**  23 IRepository**<**Astronaut**>** **\***astronauts**;**  24 IRepository**<**Planet**>** **\***planets**;**  25 IMission **\***mission**;**  26 std**::**set**<**std**::**string**>** exploredPlanets**;**  27  28 public**:**  29 Controller**(**IRepository**<**Astronaut**>** **&**astronauts**,** IRepository**<**Planet**>** **&**planets**,** IMission **&**mission**);**  30  31 std**::**string addAstronaut**(**const std**::**string **&**type**,** const std**::**string **&**astronautName**)** override**;**  32 std**::**string addPlanet**(**const std**::**string **&**planetName**,** std**::**vector**<**std**::**string**>** **&**items**)** override**;**  33 std**::**string retireAstronaut**(**const std**::**string **&**astronautName**)** override**;**  34 std**::**string explorePlanet**(**const std**::**string **&**planetName**)** override**;**  35 std**::**string report**()** override**;**  36  37 **~**Controller**()** override**;**  38 **};**  39  40 #endif // CONTROLLER\_H |

Всички членове на Controller-a са интерфейси, за да може да се осъществи loose coupling и всеки един от тях да може да бъде разширяван или подобряван без да се изисква допълнителна модификация на Controller-a, тъй като те се подават през конструктора му.

Тъй като Controller наследява IController, той имплементира всички декларирани функции от IController по следния начин (ще разделя имплементацията по функции):

* 1. **std::string addAstronaut(const std::string &type, const std::string &astronautName)**

|  |
| --- |
| 09 std**::**string Controller**::**addAstronaut**(**const std**::**string **&**type**,** const std**::**string **&**astronautName**)** **{**  10 Astronaut **\***astronaut**;**  11 std**::**string bg\_type**;**  12 **if** **(**type **==** "биолог" **||** type **==** "biologist"**)** **{**  13 bg\_type **=** "биолог"**;**  14 astronaut **=** **new** Biologist**(**astronautName**);**  15 **}** **else** **if** **(**type **==** "геодезист" **||** type **==** "geodesist"**)** **{**  16 bg\_type **=** "геодезист"**;**  17 astronaut **=** **new** Geodesist**(**astronautName**);**  18 **}** **else** **if** **(**type **==** "метеоролог" **||** type **==** "meteorologist"**)** **{**  19 bg\_type **=** "метеоролог"**;**  20 astronaut **=** **new** Meteorologist**(**astronautName**);**  21 **}** **else** **throw** std**::**invalid\_argument**(**"Няма такъв вид астронавт!\n"**);**  22  23 **this->**astronauts**->**add**(\***astronaut**);**  24  25 **return** astronautName **+** " беше добавен, длъжност - " **+** bg\_type **+** '\n'**;**  26 **}** |

Тази функция очаква низ, определящ типа на астронавта и низ с името на астронавта. След това на 10-ти ред се декларира указател към абстрактния тип Astronaut с име astronaut. Според подаденият низ за тип се преминава през проверка, за да се установи типът астронавт и след това астронавтът се създава с наследник на Astronaut и указателят astronaut се задава да сочи към този нов обект.

Ако подаденият тип не съществува функцията “хвърля” грешка със съобщение “Няма такъв вид астронавт!”.

След коректно създаване на новият астронавт, той се добавя към съответното хранилище на астронавти, създадено в main функцията (astronauts);

Функцията връща низ с името на астронавта и неговият тип (“<Име> беше добавен, длъжност - <тип>”).

* 1. **std::string addPlanet(const std::string &planetName, std::vector<std::string> &items)**

|  |
| --- |
| 28 std**::**string Controller**::**addPlanet**(**const std**::**string **&**planetName**,** std**::**vector**<**std**::**string**>** **&**items**)** **{**  29 auto planet **=** **new** Planet**(**planetName**);**  30 **for** **(**const auto **&**item **:** items**)** **{**  31 planet**->**getItems**().**push\_back**(**item**);**  32 **}**  33 **this->**planets**->**add**(\***planet**);**  34 **return** "Успешно добавена планета - " **+** planetName **+** '\n'**;**  35 **}** |

Тази функция приема име на планета и vector с всички елементи на планетата. Той се подготвя от Engine-a като се използва същия метод за разделяне по разделител ‘ ’, като при входния низ, само че този път разделителят е ‘,’.

Създава се нова планета planet всички елементи се добавят към нейния масив от елементи и тази планета се добавя към хранилището с планети planets.

Функцията връща низ “Успешно добавена планета - <Име>”.

* 1. **std::string retireAstronaut(const std::string &astronautName)**

|  |
| --- |
| 37 std**::**string Controller**::**retireAstronaut**(**const std**::**string **&**astronautName**)** **{**  38 auto astronautToRemove **=**  std**::**find\_if**(this->**astronauts**->**getModels**().**begin**(),**  39 **this->**astronauts**->**getModels**().**end**(),**  40 **[&**astronautName**](**const Astronaut **&**a**)** **{**  41 **return** astronautName **==** a**.**getName**();**  42 **});**  43 **if** **(**astronautToRemove **!=** **this->**astronauts**->**getModels**().**end**())**  44 **if** **(this->**astronauts**->**remove**(\***astronautToRemove**))**  45 **return** "Успешно пенсиониран астронавт - " **+** astronautName **+** '\n'**;**  46 **return** "Няма такъв астронавт...\n"**;**  47 **}** |

Функцията има за цел да “пенсионира” астронавт по дадено име. Реализирал съм това като използвам find\_if от STL на С++ и филтрирам астронавтите от хранилището с ламбда функция, която сравнява подаденото име с това на текущия астронавт.

Ако има съвпадение, този астронавт се записва в astronautToRemove и след това се премахва от хранилището и функцията връща низ “Успешно пенсиониран астронавт - <Име>”. Ако астронавт с такова име не е намерен, функцията връща низ “Няма такъв астронавт...”.

* 1. **std::string Controller::explorePlanet(const std::string &planetName)**

|  |
| --- |
| 49 std**::**string Controller**::**explorePlanet**(**const std**::**string **&**planetName**)** **{**  50 std**::**shuffle**(this->**astronauts**->**getModels**().**begin**(),** **this->**astronauts**->**getModels**().**end**(),**  51 std**::**mt19937**(**std**::**random\_device**()()));**  52 std**::**vector**<**Astronaut **\*>** suitableAstronauts**;**  53 **for** **(**auto **&**astronaut **:** astronauts**->**getModels**())** **{**  54 **if** **(**astronaut**.**getOxygen**()** **>** 20 **&&** **!**astronaut**.**getBag**().**isFull**())** **{**  55 suitableAstronauts**.**push\_back**(&**astronaut**);**  56 **}**  57 **}**  58 **if** **(**suitableAstronauts**.**empty**())**  59 **throw** std**::**logic\_error**(**"Няма годни астронавти за експедиция...\n"**);**  60 auto planet **=** std**::**find\_if**(this->**planets**->**getModels**().**begin**(),** **this->**planets**->**getModels**().**end**(),**  61 **[&**planetName**](**const Planet **&**p**)** **{** **return** p**.**getName**()** **==** planetName**;** **});**  62 **if** **(**planet **==** **this->**planets**->**getModels**().**end**())** **throw** std**::**invalid\_argument**(**"Няма планета с такова име!"**);**  63  64 bool killAll **=** **(**1 **+** rand**()** **%** 100**)** **>** 49**;**  65 **this->**mission**->**explore**(\***planet**,** suitableAstronauts**,** killAll**);**  66 **this->**exploredPlanets**.**insert**(**planetName**);**  67 auto deadAstronauts **=** std**::**count\_if**(**suitableAstronauts**.**begin**(),** suitableAstronauts**.**end**(),**  68 **[](**Astronaut **\***astronaut**)** **{** **return** **!**astronaut**->**canBreathe**();** **});**  69 **return** "Експедиция на планетата " **+** planetName **+** " завърши! Загинали астронавти: " **+**  70 std**::**to\_string**(**deadAstronauts**)** **+** '\n'**;**  71 **}** |

Това е функцията, в която се случва главното действие. Входният ѝ параметър е името на планетата, която ще се изследва. На ред 50 започва разместване на астронавтите в хранилището по случаен начин. За да постигна успешна случайност ползвам различните класове от библиотеката със случайни числа на C ++:

* std :: random\_device - равномерно разпределен генератор на случайни числа, който може да има достъп до хардуерно устройство във вашата система или нещо подобно /dev/random в Linux. Обикновено се използва просто за seed на псевдослучайни генератори, тъй като обикновено устройството бързо изчерпва ентропията.
* std :: mt19937 е бърз генератор на псевдослучайни числа, използващ двигателя Mersenne Twister, който според [оригиналното заглавие](http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/mt.pdf) на авторите на труда върху Mersenne Twister, също е еднакъв. Това генерира напълно произволни 32-битови или 64-битови беззнакови цели числа. Тъй като std :: random\_device се използва само за seed на този генератор, не е необходимо да е еднообразно (напр. често се сийдва генератор, използвайки текущ времеви фрагмент, който определено не е равномерно разпределен).
* std :: shuffle, според документацията,
  + “Подрежда елементите в дадения диапазон [първи, последен) така, че всяка възможна пермутация на тези елементи има еднаква вероятност за поява.”

След това преминавам към филтриране на годните астронавти, а такива са с над 20 единици кислород и с раница, която не е пълна. Годните астронавти се събират във vector suitableAstronauts. Ако няма годни астронавти, функцията “хвърля” грешка с текст “Няма годни астронавти за експедиция...”

Намира се планетата с подаденото име, отново ползвайки find\_if . Ако не е намерена такава планета се хвърля грешка текст “Няма планета с такова име!”.

На ред 64 се декларира променливата killAll, чиято цел е да се пресметне дали всички астронавти ще умрат по време на мисията. После тя се подава към функцията explore на обекта mission (explore и Mission ще бъдат разгледани в “4. Класът Mission”).

След завършването на функцията explore програмата добавя името на планетата към списък с изследваните планети, преброяват се загиналите астронавти и функцията връща низ “Експедиция на планетата <Име> завърши! Загинали астронавти: <Брой>”.

* 1. **std::string Controller::report()**

|  |
| --- |
| 73 std**::**string Controller**::**report**()** **{**  74 std**::**stringstream ss**;**  75 ss **<<** **this->**exploredPlanets**.**size**()** **<<** " планети бяха изследвани!\n"  76 **<<** "Информация за астронавтите:\n"**;**  77 **for** **(**auto **&**astronaut **:** **this->**astronauts**->**getModels**())** **{**  78 std**::**string bagItems **=**  79 std**::**accumulate**(**std**::**begin**(**astronaut**.**getBag**().**getItems**()),** std**::**end**(**astronaut**.**getBag**().**getItems**()),**  80 std**::**string**(),**  81 **[](**std**::**string **&**ss**,** std**::**string **&**s**)** **{**  82 **return** ss**.**empty**()** **?** s **:** ss**.**append**(**", "**).**append**(**s**);**  83 **});**  84 bagItems **=** **!**bagItems**.**empty**()** **?** bagItems **:** "няма"**;**  85 ss **<<** "## Име: " **<<** astronaut**.**getName**()** **<<** '\n'  86 **<<** "## Оставащ кислород: " **<<** astronaut**.**getOxygen**()** **<<** '\n'  87 **<<** "## Предмети: " **<<** bagItems **<<** '\n'**;**  88 **}**  89 ss **<<** std**::**endl**;**  90 **return** ss**.**str**().**substr**(**0**,** ss**.**str**().**length**()** **-** 1**);**  91 **}** |

Функцията report е последния фрагмент от класа Controller. Тя има за цел да изведе информация за всичко случило се до този момент. Това включва колко планети са били изследвани и информация за всички астронавти (име, оставащ кислород, предмети).

Връща форматиран низ с информацията:

“

<Брой> планети бяха изследвани!

Информация за астронавтите:

## Име: <Име>

## Оставащ кислород: <Единици>

## Предмети: <няма или предметите разделени по “, ”>

## Име: <Име>

## Оставащ кислород: <Единици>

## Предмети: <“няма” или предметите разделени по “, ”>

...

”

1. **Класът Mission**

Класът Mission е наследник на IMission, а той декларира следната функция:

|  |
| --- |
| 03 #include "planet.hpp"  04 #include "astronaut.hpp"  05 #include **<**vector**>**  06 class IMission  07 **{**  08 public**:**  09 virtual void explore**(**Planet**&,** std**::**vector**<**Astronaut**\*>&,** bool**)** **=** 0**;**  10  11 virtual **~**IMission**()** **=** **default;**  12 **};**  13 #endif // IMISSION\_H |

void explore**(**Planet**&,** std**::**vector**<**Astronaut**\*>&,** bool**)** приема планета, контейнер с астронавти и булева променлива, която указва дали мисята ще бъде пагубна или не.

Mission имплементира интерфейса по следния начин:

|  |
| --- |
| 01 #include "mission.hpp"  02 #ifdef \_\_EMSCRIPTEN\_\_  03 #include **<**emscripten**/**bind**.**h**>**  04 **using** **namespace** emscripten**;**  05 #endif  06 void Mission**::**explore**(**Planet **&**planet**,** std**::**vector**<**Astronaut **\*>** **&**astronauts**,** bool killAll**)** **{**  07 **for(**auto **&**astronaut**:** astronauts**)**  08 **{**  09 **if(!**astronaut**->**canBreathe**()** **||** astronaut**->**getBag**().**isFull**())**  10 **{**  11 **continue;**  12 **}**  13 **while** **(**astronaut**->**canBreathe**())** **{**  14 **if(**planet**.**getItems**().**empty**()** **||** astronaut**->**getBag**().**isFull**())** **break;**  15 auto item **=** planet**.**getItems**().**front**();**  16 astronaut**->**getBag**().**getItems**().**push\_back**(**item**);** //добавя предмета от планетата в раницата на текущия астронавт  17 int chance **=** rand**()%**100 **+** 1**;** //случайна променлива за взимането на кислород, 33% шанс да намали кислорода 2 пъти  18 **if** **(**chance**%**3 **<** 0.00001**)**  19 astronaut**->**breathe**();**  20 astronaut**->**breathe**();**  21 auto planetItems **=** **&**planet**.**getItems**();**  22 planetItems**->**erase**(**std**::**remove**(**planetItems**->**begin**(),** planetItems**->**end**(),** item**),** planetItems**->**end**());** //премахване на предмета от планетата  23 **}**  24 **if(**killAll**)** //проверка дали е необходимо дали всички астронавти от експедицията да бъдат убити  25 **{**  26 astronaut**->**name **+=** " - убит/изчезнал"**;**  27 astronaut**->**oxygen **=** 0**;**  28 **}**  29 **}**  30 **}**  31  32 Mission**::~**Mission**()** **=** **default;**  33  34 #ifdef \_\_EMSCRIPTEN\_\_  35 EMSCRIPTEN\_BINDINGS**(**mission\_class**){**  36 class\_**<**Mission**>(**"Mission"**)**  37 **.**constructor**<>()**  38 **;**  39 **}**  40 #endif |

Започва последователно итериране на контейнера с астронавти. Взима се всеки един астронавт и се проверява дали има достатъчно кислород, за да продължи и също така дали има място в раницата си. Ако едно от двете неща не е изпълнено, се преминава към следващия астронавт. След това се влиза в while цикъл с условие докато астронавтът може да диша. Проверява се дали планетата има елементи и дали астронавтът има място в раницата си и ако едно от тези неща не е изпълнено обиколката на този астронавт приключва. Ако горните условия са изпълнени, астронавтът намира елемент и го прибира в раницата си. По време на самото действие се изчислява дали е имало усложнения при взимането и в зависимост от променливата chance с 33% вероятност да е имало се налага астронавтът да изхаби двойно повече кислород от нормалното количество за типа си (типовете астронавти ще бъдат рагледани в “5. Класът Astronaut”). След това предметът се премахва от планетата ползвайки remove от STL. С това while цикълът приключва.

Преминава се през проверка дали е писано на всички астронавти да загинат и ако killAll е true, то към името на текущия астронавт се добавя “ - убит/изчезнал” и неговият кислород се приравнява на 0. След това се преминава към следващият астронавт.

След итериране през всички астронавти функцията приключва своето изпълнение.

1. **Класът Astronaut**

Класът е абстрактен и неговата роля е да въведе основна логика за наследяване, валидиране на входни данни и декларации за “договор” между наследниците и него.

Деклариран е по следния начин:

|  |
| --- |
| 09 class Astronaut **{**  10 friend class Mission**;**  11 std**::**string name**;**  12 double oxygen**;**  13 Backpack bag**;**  14  15 void setName**(**const std**::**string **&**value**);**  16 protected**:**  17 Astronaut**(**const std**::**string **&**name**,** double oxygen**);**  18 void setOxygen**(**const double **&**value**);**  19 public**:**  20 const std**::**string **&**getName**()** const**;**  21 double getOxygen**()** const**;**  22 bool canBreathe**()** const**;**  23 Backpack **&**getBag**();**  24 virtual void breathe**();**  25 bool **operator==(**const Astronaut **&**a**);**  26  27 virtual **~**Astronaut**();**  28 **};** |

Класът декларира, че всеки наследник трябва да има членове:

- std**::**string name - име

- double oxygen - кислород

- Backpack bag - раница

Функции за задаване на данни и извличане на данни/аксесори и мутатори:

- void setName**(**const std**::**string **&**value**)** - setter за име

- void setOxygen**(**const double **&**value**)** - setter за кислород

- const std**::**string **&**getName**()** const - getter за име

- double getOxygen**()** const - getter за кислород

- bool canBreathe**()** const - калкулиран getter за информация дали астронавтът има останал кислород

- Backpack **&**getBag**()** - фунцкия даваща достъп до раницата на астронавта чрез референция

Функции за действия и операции:

- virtual void breathe**()** - функция за дишане

- bool **operator==(**const Astronaut **&**a**)** - пренаписан оператор за сравнение на два класа от тип Astronaut или негов наследник

След това Astronaut имплементира някои от декларираните функции, като оставя breathe() виртуална, за да може да бъде променяна, че да се осъществят различните начини на дишане на различните астронавти.

|  |
| --- |
| 01 #include "astronaut.hpp"  02  03 void Astronaut**::**setName**(**const std**::**string **&**value**)** **{**  04 **if** **(**value**.**empty**()** **||** value **==** " "**)** **{**  05 **throw** std**::**invalid\_argument**(**"Името на астронавт не може да бъде празно!\n"**);**  06 **}**  07 **this->**name **=** value**;**  08 **}**  09  10 Astronaut**::**Astronaut**(**const std**::**string **&**name**,** const double oxygen**)** **{**  11 **this->**setName**(**name**);**  12 **this->**setOxygen**(**oxygen**);**  13 **}**  14  15 void Astronaut**::**setOxygen**(**const double **&**value**)** **{**  16 **if** **(**value **<** 0**)** **{**  17 **throw** std**::**invalid\_argument**(**"Не може да се зададе негативна стойност за кислород!\n"**);**  18 **}**  19 **this->**oxygen **=** value**;**  20 **}**  21  22 const std**::**string **&**Astronaut**::**getName**()** const **{**  23 **return** **this->**name**;**  24 **}**  25  26 double Astronaut**::**getOxygen**()** const **{**  27 **return** **this->**oxygen**;**  28 **}**  29  30 bool Astronaut**::**canBreathe**()** const **{**  31 **return** **this->**oxygen **>** 0**;**  32 **}**  33  34 Backpack **&**Astronaut**::**getBag**()** **{**  35 **return** **this->**bag**;**  36 **}**  37  38 void Astronaut**::**breathe**()** **{**  39 **this->**setOxygen**(this->**oxygen **-** 10 **<** 0 **?** 0 **:** **this->**oxygen **-** 10**);**  40 **}**  41  42 bool Astronaut**::operator==(**const Astronaut **&**a**)** **{** **return** **this->**name **==** a**.**getName**();** **}**  43  44 Astronaut**::~**Astronaut**()** **=** **default;** |

Чрез декларираните setter-и се извършва валидация.

* setName проверява дали подаденият аргумент за име е празен string или whitespace. Ако валидацията не мине както трябва, функцията “хвърля” грешка със съобщение “Името на астронавт не може да бъде празно!”.
* setOxygen проверява дали подаденият аргумент за кислород е по-малък по стойност или равен на 0. Ако проверката не мине, функцията “хвърля” грешка със съобщение “Не може да се зададе негативна стойност за кислород!”.

Всички getter-и дават директен достъп до членовете, за които отговарят.

* canBreathe е калкулиран getter затова той пресмята дали полето oxygen e по-голямо от 0. Ако е по-голямо от 0 връща true, ако не е - false.

Функцията breathe се имплементира като използва setOxygen, за да зададе стойност с 10 единици по-ниска от сегашната или ако нивото падне под 0, да зададе 0. Както споменах, това е виртуална функция, така че нейното действие може да бъде променено от класовете наследници.

Операторът == се пренаписва така, че да връща true, когато дадени две инстанции на Astronaut или наследник имат едни и същи полета name (когато имената им са едни и същи), false - в останалите случаи.

**Източници:**

* Robert C. Martin (Uncle Bob), [Design Principles and Design Patterns](https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf) *//SOLID*
* Makoto Matsumoto, Takuji Nishimura, [Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator](http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ARTICLES/mt.pdf) //explorePlanet
* C++ Reference Documentation, [Pseudo-random number generation](https://en.cppreference.com/w/cpp/numeric/random) //explorePlanet