Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Уфимский государственный авиационный технический университет»

Факультет информатики и робототехники

Кафедра Вычислительной математики и кибернетики

**Лабораторная работа №2**

По дисциплине «Объектно-ориентированное программирование»

«Объекты и Классы»

Выполнил: студент группы ПРО-222

Вальшин Руслан Ильдусович

УФА 2020

1. **Цель работы:**

Определение и реализация тестовых классов, и написание программы, иллюстрирующей их использование. Разобраться и понимать, что будет если:

• в классе-предке есть конструктор и деструктор, а в классе-потомке нет ни того, ни того; что происходит при создании предка, при создании потомка?

• в классе-предке и в классе потомке есть и конструкторы и деструкторы: в каком порядке они вызываются при создании и удалении объекта?

• в переменную-указатель на базовый класс поместить объект-потомок: какие методы можно вызывать, какие нет?

1. **Задачи:**

* Определения
  + классов
  + атрибутов, методов
  + классов-наследников
  + конструкторов (обязательно несколько: без параметров, с параметрами, с параметром-объектом того же класса), деструкторов
  + доступности атрибутов и методов (показать, как влияют модификаторы private, protected, public)
* Реализация
  + методов объектов
  + реализация методов сразу в определении или после определения
  + конструкторов, деструкторов
* Создание и использование (обращение к атрибутам, вызов методов)
  + статически создаваемых объектов («MyClass obj;»)
  + динамически создаваемых объектов («MyClass \*obj = new MyClass();»)
  + объектов с помощью различных конструкторов (**у каждого** создаваемого объекта должны быть: конструктор без параметров, с параметрами, с параметром-объектом того же класса – конструктор копирования)
  + помещение объектов в переменные различных типов (объяснять, чем отличается MyBase \* obj = new MyBase() от MyBase \* obj = new MyDeriv())
  + объектов классов-наследников (проверить и продемонстрировать, какие конструкторы классов при этом вызываются)
  + композиция объектов: атрибутом одного объекта класса A является указатель на другой объект класса B, создаваемый в конструкторе класса A и уничтожаемый в деструкторе класса A; композируемые классы должны определяться отдельно (не один в другом).
* Уничтожение
  + статически созданных объектов
  + динамически созданных объектов
  + объектов классов-наследников (проверить и продемонстрировать, какие деструкторы классов при этом вызываются)

1. **Ход работы**:

Определение класса CelestialBody (Небесное Тело). У небесного тела как объекта есть атрибуты: масса, объём и период вращения вокруг собственной оси (1 день). У класса есть метод message(), который сообщает о природе этого объекта. При этом метод виртуальный. Когда будут созданы наследники, этот метод перезапишется.

У класса есть 3 конструктора – простой, с параметрами и конструктор копирования. Деструктор уничтожает объект класса.

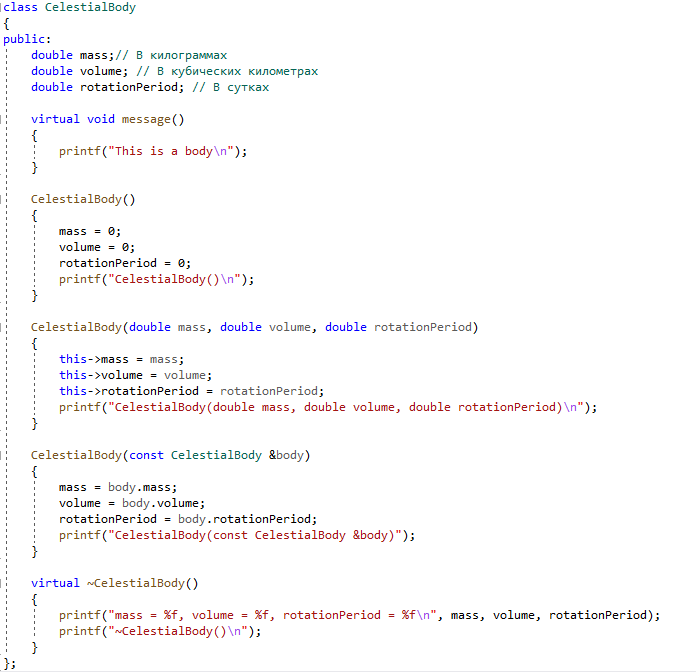


Рисунок 1 – Класс CelestialBody

Определяется класс Galaxy с защищённым атрибутом galaxyName. Чтобы получить имя галактики, можно воспользоваться методом getGalaxyName(). Можно узнать о природе объекта через message(). У класса также есть все три основных конструктора и деструктор.



Рисунок 2 – Класс Galaxy

Определяется класс PlanetarySystem, наследуемый от Galaxy (Планетарная система находится в какой-то галактике). Методом getPlanetarySystem() можно получить имя системы. message() сообщит о природе объекта. Есть 3 основных конструктора и деструктор.

Рисунок 3 – Класс PlanetarySystem

Определяется класс Moon, наследующий класс CelestialBody. У класса есть атрибут moonName, метод message, сообщающий о природе объекта и перезаписывающий метод предка. Есть 3 конструктора и деструктор. Конструкторы с параметрами вызывают соответствующие конструкторы предка.



Рисунок 4 – Класс Moon

Определяется класс Planet, наследующий два класса PlanetarySystem и CelestialBody. У класса есть закрытый атрибут numberOfMoons, список из объектов другого класса moon, метод addMoon, который копирует переданный объект в атрибут moon, метод message, 3 конструктор, деструктор (удаляет в том числе и содержащиеся луны).

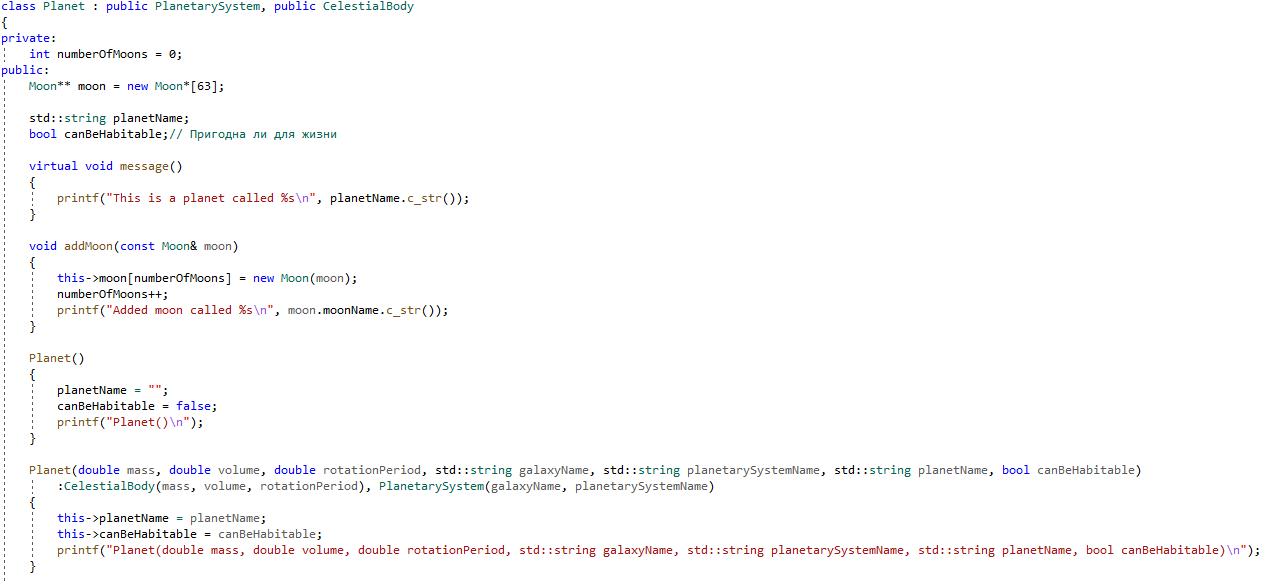
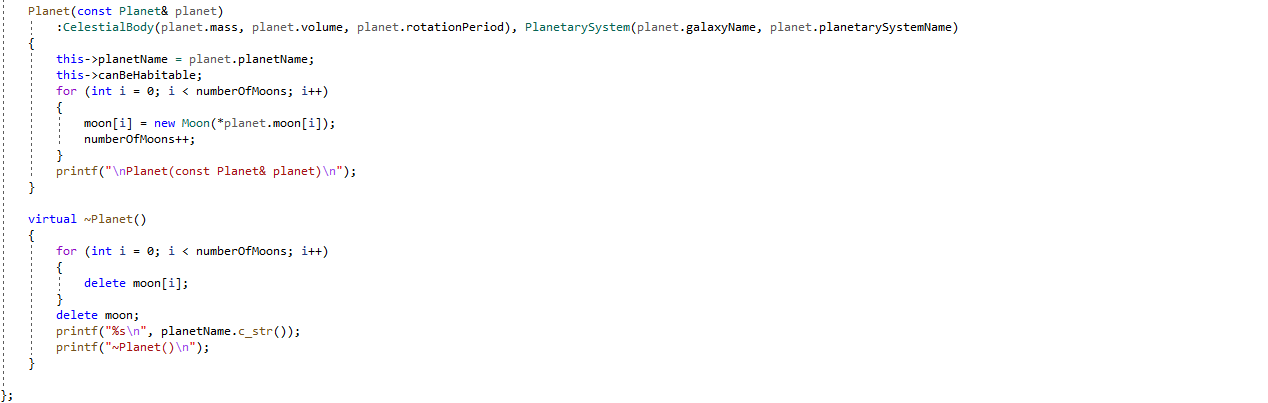
 

Рисунок 5 – Класс Planet

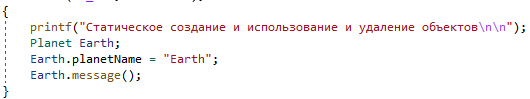


Рисунок 6 - Статическое создание объекта

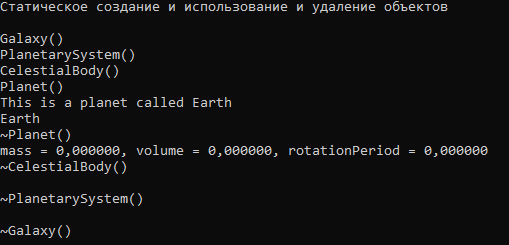


Рисунок 7 - Отладочный вывод статического создания

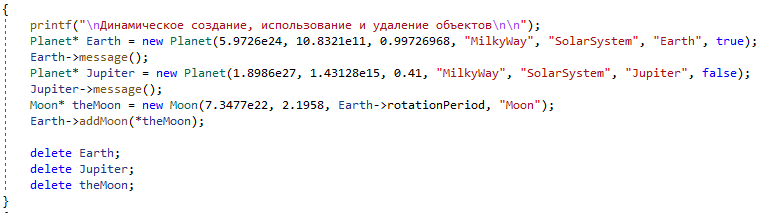


Рисунок 8 - Динамическое создание объекта

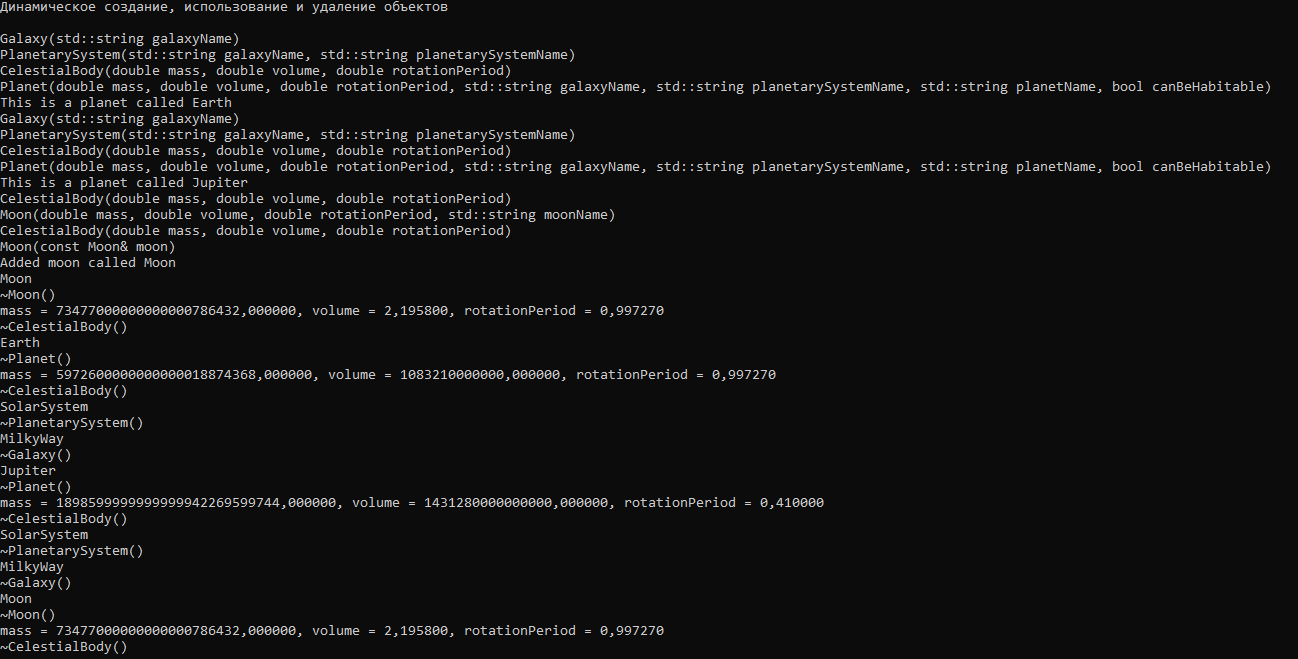


Рисунок 9 - Отладочный вывод динамического создания объекта

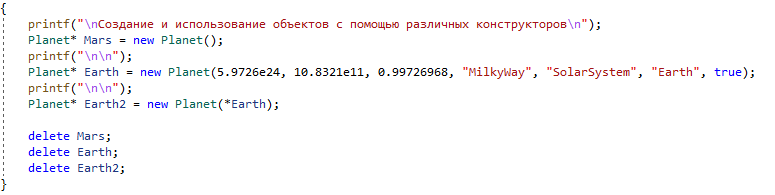


Рисунок 10 - Создание и использование объектов с помощью различных конструкторов

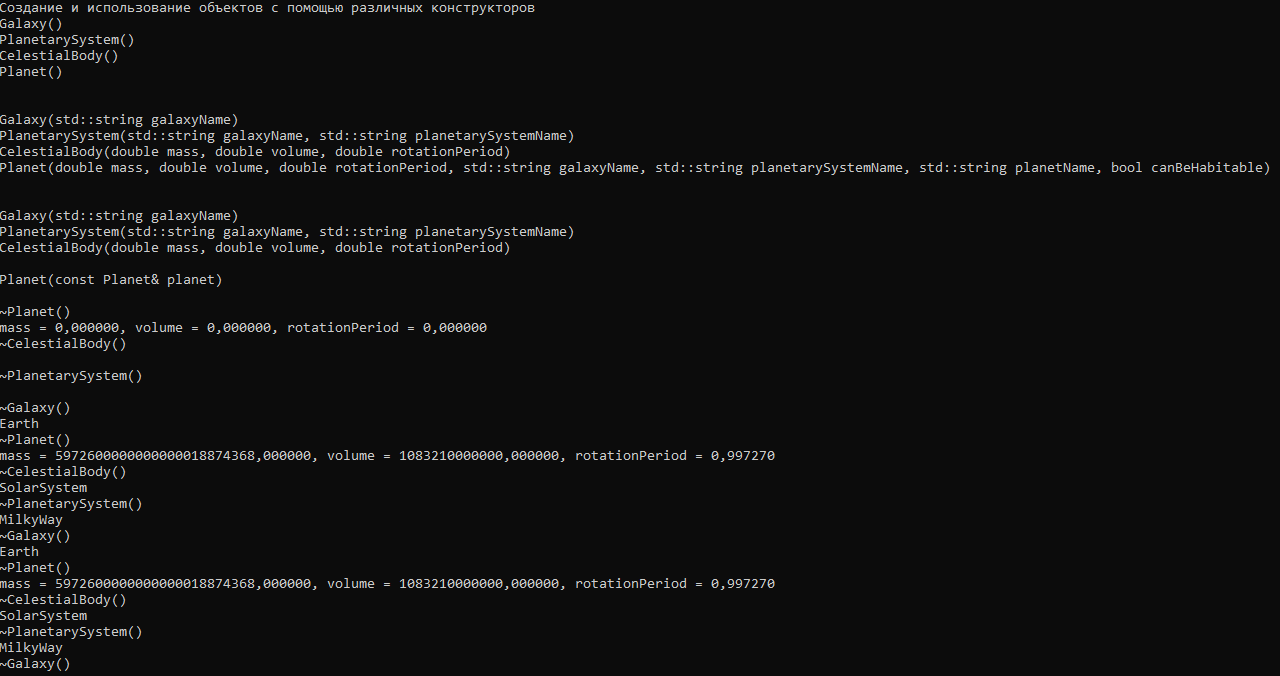


Рисунок 11 - Отладочный вывод создания и использования объектов с помощью различных конструкторов

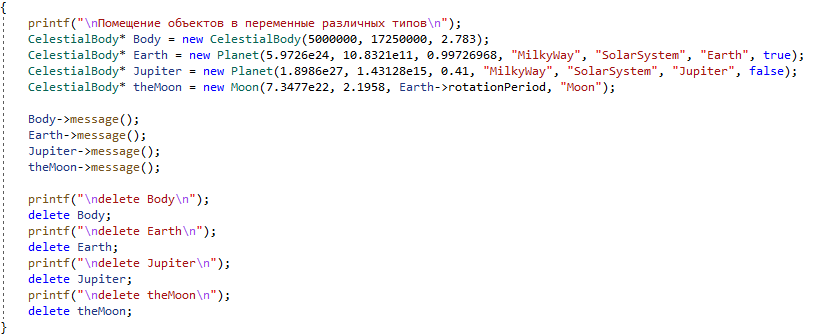


Рисунок 12 - Помещение объектов в различные типы данных

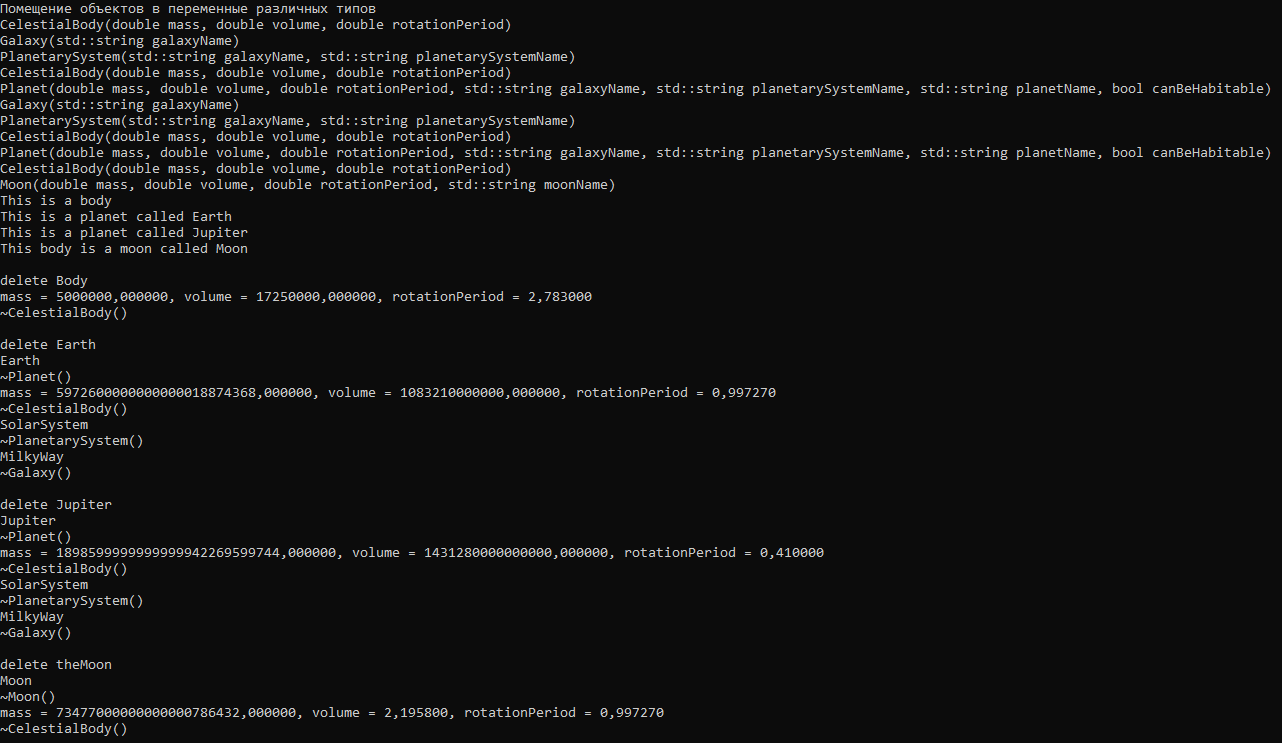


Рисунок 13 – Отладочный вывод помещения объектов в различные типы данных

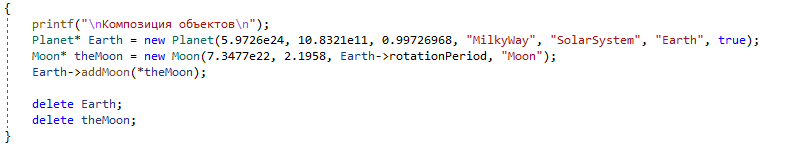


Рисунок 14 - Композиция объектов

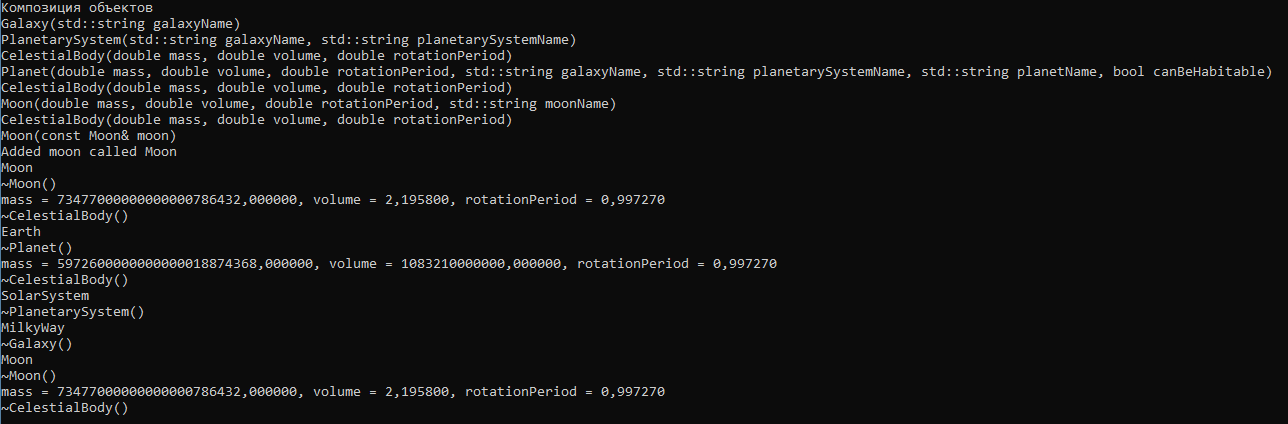
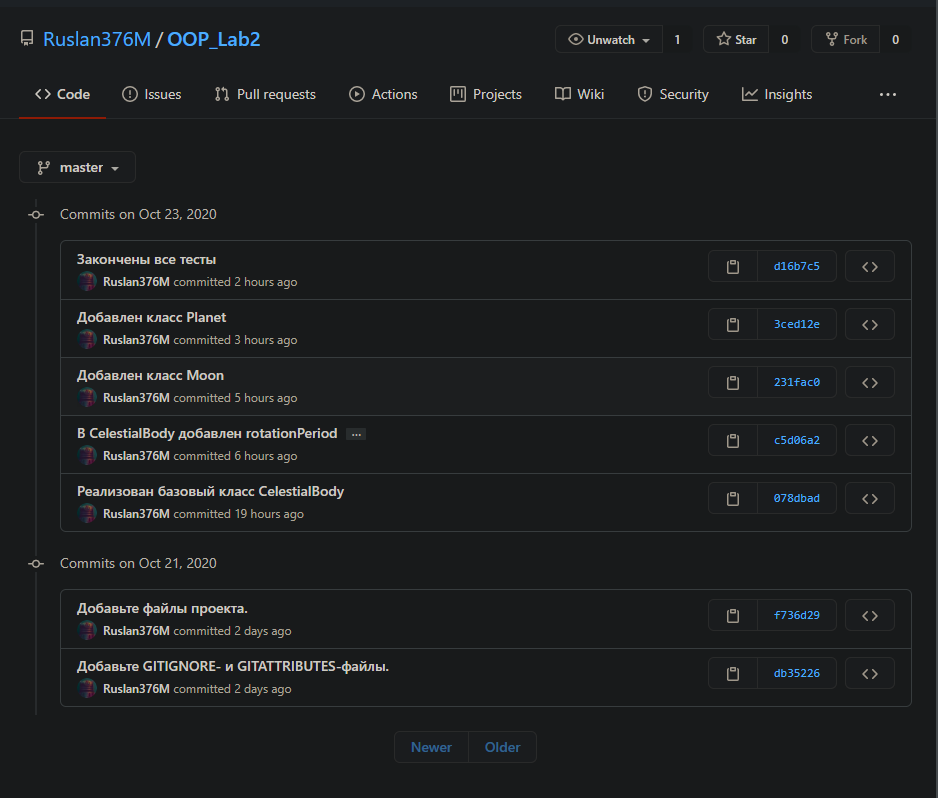


Рисунок 15 - Отладочный вывод композиции объектов

1. **Вывод:** В ходе этой работы я научился определять классы, атрибуты и методы, классы-наследники, конструкторы трёх различных типов, задавать модификаторы доступа; создавать и использовать статические и динамические объекты, объекты с помощью различных конструкторов, помещать объекты в переменные различных типов, объекты классов-наследников, композицию объектов, уничтожать статически и динамически созданные объекты и объекты классов-наследников.

Приложение 1. Ссылка на GitHub репозиторий и скриншот истории коммитов.

<https://github.com/Ruslan376M/OOP_Lab2>



Приложение 2. Листинг программы.

#include <iostream>

#include <locale.h>

class CelestialBody

{

public:

double mass;// В килограммах

double volume; // В кубических километрах

double rotationPeriod; // В сутках

virtual void message()

{

printf("This is a body\n");

}

CelestialBody()

{

mass = 0;

volume = 0;

rotationPeriod = 0;

printf("CelestialBody()\n");

}

CelestialBody(double mass, double volume, double rotationPeriod)

{

this->mass = mass;

this->volume = volume;

this->rotationPeriod = rotationPeriod;

printf("CelestialBody(double mass, double volume, double rotationPeriod)\n");

}

CelestialBody(const CelestialBody &body)

{

mass = body.mass;

volume = body.volume;

rotationPeriod = body.rotationPeriod;

printf("CelestialBody(const CelestialBody &body)");

}

virtual ~CelestialBody()

{

printf("mass = %f, volume = %f, rotationPeriod = %f\n", mass, volume, rotationPeriod);

printf("~CelestialBody()\n");

}

};

class Galaxy

{

protected:

std::string galaxyName;

public:

std::string getGalaxyName()

{

return galaxyName;

}

void message()

{

printf("This body belongs to %s Galaxy\n", galaxyName.c\_str());

}

Galaxy()

{

galaxyName = "";

printf("Galaxy()\n");

}

Galaxy(std::string galaxyName)

{

this->galaxyName = galaxyName;

printf("Galaxy(std::string galaxyName)\n");

}

Galaxy(const Galaxy& galaxy)

{

galaxyName = galaxy.galaxyName;

printf("Galaxy(const Galaxy& galaxy)\n");

}

virtual ~Galaxy()

{

printf("%s\n", galaxyName.c\_str());

printf("~Galaxy()\n");

}

};

class PlanetarySystem: public Galaxy

{

protected:

std::string planetarySystemName;

public:

std::string getPlanetarySystemName()

{

return planetarySystemName;

}

void message()

{

printf("This body belongs to %s System\n", planetarySystemName.c\_str());

}

PlanetarySystem()

{

planetarySystemName = "";

printf("PlanetarySystem()\n");

}

PlanetarySystem(std::string galaxyName, std::string planetarySystemName)

:Galaxy(galaxyName)

{

this->planetarySystemName = planetarySystemName;

printf("PlanetarySystem(std::string galaxyName, std::string planetarySystemName)\n");

}

PlanetarySystem(const PlanetarySystem& planetarySystem)

:Galaxy(planetarySystem.galaxyName)

{

planetarySystemName = planetarySystem.planetarySystemName;

printf("PlanetarySystem(const PlanetarySystem& planetarySystem)\n");

}

virtual ~PlanetarySystem()

{

printf("%s\n", planetarySystemName.c\_str());

printf("~PlanetarySystem()\n");

}

};

class Moon : public CelestialBody

{

public:

std::string moonName;

Moon()

{

moonName = "";

printf("Moon()\n");

}

virtual void message()

{

printf("This body is а moon called %s\n", moonName.c\_str());

}

Moon(double mass, double volume, double rotationPeriod, std::string moonName)

:CelestialBody(mass, volume, rotationPeriod)

{

this->moonName = moonName;

printf("Moon(double mass, double volume, double rotationPeriod, std::string moonName)\n");

}

Moon(const Moon& moon)

:CelestialBody(moon.mass, moon.volume, moon.rotationPeriod)

{

moonName = moon.moonName;

printf("Moon(const Moon& moon)\n");

}

virtual ~Moon()

{

printf("%s\n", moonName.c\_str());

printf("~Moon()\n");

}

};

class Planet : public PlanetarySystem, public CelestialBody

{

private:

int numberOfMoons = 0;

public:

Moon\*\* moon = new Moon\*[63];

std::string planetName;

bool canBeHabitable;// Пригодна ли для жизни

virtual void message()

{

printf("This is а planet called %s\n", planetName.c\_str());

}

void addMoon(const Moon& moon)

{

this->moon[numberOfMoons] = new Moon(moon);

numberOfMoons++;

printf("Added moon called %s\n", moon.moonName.c\_str());

}

Planet()

{

planetName = "";

canBeHabitable = false;

printf("Planet()\n");

}

Planet(double mass, double volume, double rotationPeriod, std::string galaxyName, std::string planetarySystemName, std::string planetName, bool canBeHabitable)

:CelestialBody(mass, volume, rotationPeriod), PlanetarySystem(galaxyName, planetarySystemName)

{

this->planetName = planetName;

this->canBeHabitable = canBeHabitable;

printf("Planet(double mass, double volume, double rotationPeriod, std::string galaxyName, std::string planetarySystemName, std::string planetName, bool canBeHabitable)\n");

}

Planet(const Planet& planet)

:CelestialBody(planet.mass, planet.volume, planet.rotationPeriod), PlanetarySystem(planet.galaxyName, planet.planetarySystemName)

{

this->planetName = planet.planetName;

this->canBeHabitable;

for (int i = 0; i < numberOfMoons; i++)

{

moon[i] = new Moon(\*planet.moon[i]);

numberOfMoons++;

}

printf("\nPlanet(const Planet& planet)\n");

}

virtual ~Planet()

{

for (int i = 0; i < numberOfMoons; i++)

{

delete moon[i];

}

delete moon;

printf("%s\n", planetName.c\_str());

printf("~Planet()\n");

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

{

printf("Статическое создание и использование и удаление объектов\n\n");

Planet Earth;

Earth.planetName = "Earth";

Earth.message();

}

{

printf("\nДинамическое создание, использование и удаление объектов\n\n");

Planet\* Earth = new Planet(5.9726e24, 10.8321e11, 0.99726968, "MilkyWay", "SolarSystem", "Earth", true);

Earth->message();

Planet\* Jupiter = new Planet(1.8986e27, 1.43128e15, 0.41, "MilkyWay", "SolarSystem", "Jupiter", false);

Jupiter->message();

Moon\* theMoon = new Moon(7.3477e22, 2.1958, Earth->rotationPeriod, "Moon");

Earth->addMoon(\*theMoon);

delete Earth;

delete Jupiter;

delete theMoon;

}

{

printf("\nСоздание и использование объектов с помощью различных конструкторов\n");

Planet\* Mars = new Planet();

printf("\n\n");

Planet\* Earth = new Planet(5.9726e24, 10.8321e11, 0.99726968, "MilkyWay", "SolarSystem", "Earth", true);

printf("\n\n");

Planet\* Earth2 = new Planet(\*Earth);

delete Mars;

delete Earth;

delete Earth2;

}

{

printf("\nПомещение объектов в переменные различных типов\n");

CelestialBody\* Body = new CelestialBody(5000000, 17250000, 2.783);

CelestialBody\* Earth = new Planet(5.9726e24, 10.8321e11, 0.99726968, "MilkyWay", "SolarSystem", "Earth", true);

CelestialBody\* Jupiter = new Planet(1.8986e27, 1.43128e15, 0.41, "MilkyWay", "SolarSystem", "Jupiter", false);

CelestialBody\* theMoon = new Moon(7.3477e22, 2.1958, Earth->rotationPeriod, "Moon");

Body->message();

Earth->message();

Jupiter->message();

theMoon->message();

printf("\ndelete Body\n");

delete Body;

printf("\ndelete Earth\n");

delete Earth;

printf("\ndelete Jupiter\n");

delete Jupiter;

printf("\ndelete theMoon\n");

delete theMoon;

}

{

printf("\nКомпозиция объектов\n");

Planet\* Earth = new Planet(5.9726e24, 10.8321e11, 0.99726968, "MilkyWay", "SolarSystem", "Earth", true);

Moon\* theMoon = new Moon(7.3477e22, 2.1958, Earth->rotationPeriod, "Moon");

Earth->addMoon(\*theMoon);

delete Earth;

delete theMoon;

}

}