COMP123 – Abstract Planets

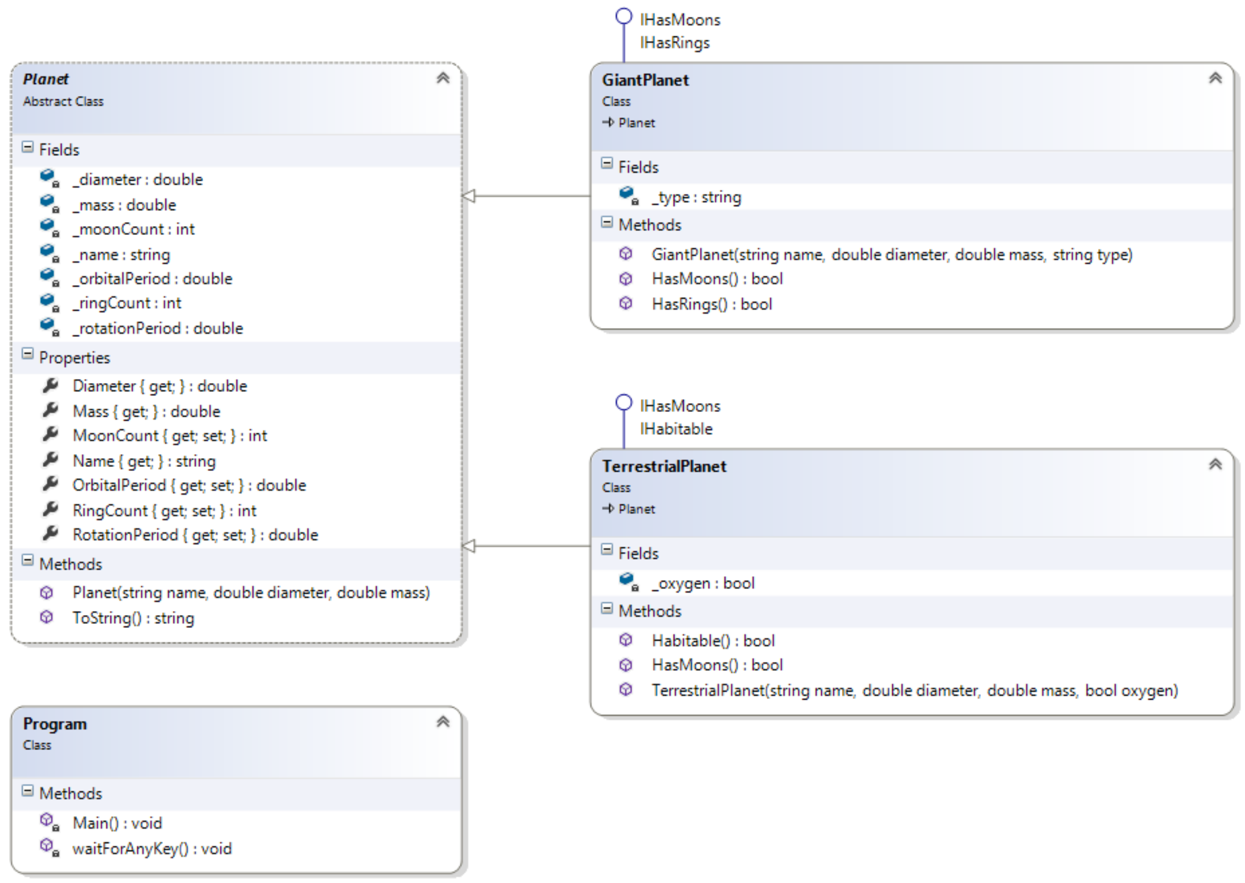
Assignment 4
Abstract Planets

Due Week #11 (Friday July 28, 2017) @ midnight.

Value 10%

Abstract Planets **Maximum Mark: 68**

Overview: Build and Implement the following classes and interfaces. This is a solo assignment.



Instructions :

(50 Marks: Functionality, 8 Marks: Program Structure, 6 Marks: Internal Documentation, 4 Marks: Version Control)

1. The **Abstract** Class ***Planet*** must include the following **Properties** and **Methods**: **(24 Marks: Functionality):**
    a. *Private* Instance Variables: **_diameter (double), _mass (double), _moonCount (int), _name (string), _orbitalPeriod (double), _ringCount (int), _rotationPeriod (double)** (7 Marks: Functionality).
    b. *Public Properties:* **Diameter** (Read Only), **Mass** (Read Only), **MoonCount**, **Name** (Read Only), **OrbitalPeriod**, **RingCount**, and **RotationPeriod** (7 Marks: Functionality).
    c. The **constructor Method** should take **name**, **diameter** and **mass** as local variables and set the related instance variables (**_name**, **_diameter** and **_mass**) to their values (6 Marks: Functionality).
    d. **Override** the inherited **`ToString()`** method so that it outputs **Name**, **Diameter** and **Mass** to the Console (4 Marks: Functionality).
2. The class **GiantPlanet** that is a subclass of the **Planet** Abstract class. The class will also implement both the **IHasMoons** and **IHasRings** interfaces: **(8 Marks: Functionality):**
    a. *Private Instance Variables:* **_type** (string) which will hold a string that describes the type of Giant Planet (either "Gas" or "Ice") (2 Marks: Functionality)
    b. The constructor Method should take all the parameters of the Planet base class as well as **type** as a local variable and set the related instance variable (**_type**) to its value (2 Marks: Functionality).
    c. Implement the **HasMoons** method that will return **true** if the **MoonCount** property is greater than zero (2 Marks: Functionality).
    d. Implement the **HasRings** method that will return **true** if the **RingCount** property is greater than zero (2 Marks: Functionality).
3. The class **TerrestrialPlanet** that is a subclass of the **Planet** Abstract class. The class will also implement both the **IHasMoons** and **IHabitable** interfaces: **(7 Marks: Functionality):**
    a. *Private Instance Variables:* _oxygen (bool) (1 Marks: Functionality)
    b. The constructor Method should take all the parameters of the Planet base class as well as **oxygen** as a local variable and set the related instance variable (**_oxygen**) to its value (2 Marks: Functionality).
    c. Implement the **HasMoons** method that will return **true** if the **MoonCount** property is greater than zero (2 Marks: Functionality).
    d. Implement the **Habitable** method that will return **true** if the **oxygen** instance variable is set to **true** (2 Marks: Functionality).
4. The Interface **IHasMoons** which will include a method header **HasMoons** that returns a **bool** data type **(2 Marks: Functionality)**.
5. The Interface **IHasRings** which will include a method header **HasRings** that returns a **bool** data type **(2 Marks: Functionality)**.

6. The Interface **IHabitable** which will include a method header **Habitable** that returns a **bool** data type **(2 Marks: Functionality)**.
7. The Main method of your driver class will be structured as follows **(5 Marks: Functionality):**
    a. The **Main** method of your driver class (**Program**), implements the **GiantPlanet** class by creating a new **giantPlanet** object (1 Mark: Functionality).
    b. Have the **giantPlanet** object call its *public* `ToString()` method to display planet's stats to the console (1 Mark: Functionality).
    c. The **Main** method of your driver class (**Program**), will also implement the **TerrestrialPlanet** class by creating a new **terrestrialPlanet** object (1 Mark: Functionality).
    d. Have the **terrestrialPlanet** object call its *public* `ToString()` method to display planet's stats to the console (1 Mark: Functionality).
    e. Implement another Method of your driver class (**Program**), **WaitForAnyKey**, that reads the console for any key press before the console is closed (1 Mark: Functionality).
8. Program Structure **(8 Marks: Program Structure):**
    a. Your "Driver" class will be named **Program.** You will use this class to create an object instance of the **GiantPlanet** and **TerrestrialPlanet** classes (1 Mark: Program Structure).
    b. Your **Abstract** Planet class will be named **Planet**. This class will be contained in a separate file named **Planet.cs** (1 Marks: Program Structure).
    c. Your Giant Planet class will be named **GiantPlanet**. This class will be contained in a separate file named **GiantPlanet.cs** (1 Marks: Program Structure).
    d. Your Terrestrial Planet class will be named **TerrestrialPlanet**. This class will be contained in a separate file named **TerrestrialPlanet.cs** (1 Marks: Program Structure).
    e. You will need **three** interface classes for this assignment: IHasMoons, IHasRings and IHabitable. Each interface will be contained in separate files named **IHasMoons.cs**, **IHasRings.cs** and **IHabitable.cs** respectively (3 Marks: Program Structure).
    f. Ensure all *private* class member variables (instance variables) use an underscore character (**_**) at the beginning of the identifier name to signify that they are private (or protected) (1 Mark: Program Structure).
9. Include Internal Documentation for your program **(6 Marks: Internal Documentation):**
    a. Ensure you include a program header that indicates: The Author's name, Author's student number, Date last Modified, Program description, Revision History (4 Marks: Documentation).
    b. Ensure your program uses contextual variable names that help make the program human-readable (2 Marks: Documentation).
10. Share your files on **GitHub** to demonstrate Version Control Best Practices **(4 Marks: Version Control).**
    a. Your repository must include **your code** and be well structured (2 Marks: Version Control).
    b. Your repository must include **commits** that demonstrate the project being updated at different stages of development – each time a major change is implemented (2 Marks: Version Control).

SUBMITTING YOUR WORK

Your submission should include:

1. A zip archive of your project files
2. A link to your project files on GitHub.

Feature	Description	Marks
Functionality	The program's deliverables are all met and the program functions as it should. No errors appear as a result of execution. User Input does not crash the program.	50
Program Structure	Your main "driver" class is named Program and it creates objects that are defined in other classes. All other classes are contained in their own files. Your classes use **public** properties and related **private** member variables wherever possible.	8
Internal Documentation	A program header is present and includes the name of the program, the name of the student, student number, date last modified, a short revision history and a short description of the program. All methods and classes include headers that describe their functionality and scope and follow commenting best practices. Inline comments are used to indicate code function where appropriate. Variable names are contextual wherever possible.	6
Version Control	GitHub commit history demonstrating regular updates.	4
Total		**68**

This assignment is weighted **10%** of your total mark for this course.

Late submissions:

- 20% deducted for each additional day.

External code (e.g. from the internet or other sources) can be used for student submissions within the following parameters:

1. The code source (i.e. where you got the code and who wrote it) must be cited in your internal documentation.
2. It encompasses a maximum of 10% of your code (any more will be considered cheating).
3. You must understand any code you use and include documentation (comments) around the code that explains its function.
4. You must get written approval from me via email.