Planetary system module

Planetary system.

Planetary system can store multiple Planets. There are also spaceships.

Planets should be created when planetary system is created. Planetary system has 2 constructors:

- Default constructor (without parameters) which creates all 8 Planets.
- Constructor with *int* type parameter which shows the number of Planets to create. Minimum value is 3.

For example: *new PlanetarySystem(3)* creates a planetary system with first 3 planets (*Mercury, Venus, Earth*)

Planetary system has a method for printing out info about all the planets and spaceships in the system (calls Planet's info methods for all planets in the system).

Planets

All planets have the following properties:

- Name
- Weight
- Length of day in hours
- Length of year in earth years
- Length of year in local solar days
- Number of spaceship landing platforms
- List of spaceships currently on that planet

A year is defined as the time it takes a planet to complete one revolution of the Sun, for Earth this is just over 365 days. This is also known as the orbital period. The the length of each planet's year correlates with its distance from the Sun.

In the table below some values are missing: calculate them when planets are created! Round them 2 places. (Format: 0.00) For calculations use the knowledge that 1 earth year = 365*24h. You find the example values for Jupiter on the next page.

	Planet's name	Length of day (*ööpäev*) in hours	Length of year (in earth years)	Length of year (in local solar days)	Weight of the planet (10^{24}kg)	Number of spaceship landing platforms
1.	Mercury	4222.6		0.498	0.330	1
2.	Venus	2802.0	0.62		4.87	2
3.	Earth	24	1	365	5.97	6
4.	Mars	24.7	1.88		0.642	2
5.	Jupiter	9.9	11.86		1898	2
6.	Saturn	10.7		24491	586	1
7.	Uranus	17.2		42786.49	86.8	0
8.	Neptune	16.1	164.8		102	1

$$Y_{Solar} = \frac{365 * 24 * Y_{Earth}}{D_{Length}}$$

All planets have:

- A method for printing info which prints:
  - Planets name
  - Size in correspondance to Earth's size (how many times smaller or bigger than Earth)
  - Length of day in hours
  - Length of year in earth years
  - Length of year in local solar days

  Example for Jupiter:

  „Planet Jupiter is 319 times bigger than Earth. Its day is 9.9 hours. Its year is 11.86 Earth years and 10475.8 solar days."

- A method for printing out spaceships info which prints:
  - All landing platforms on that planet and the name of the vehicle in the platform (if there are any).

  Example: *PrintSpaceshipInfo()->*
  Landing platform 1: empty
  Landing platform 2: Moondriver

Spaceship:

All spaceships have a name. Spaceships are initially on planet Earth. Spaceships can travel to other planets. There are two types of spaceships:

- Simple spaceships: they can travel to the planet next to them (one planet at a time).

  Example: *from Earth to Venus or Mars*

- Advanced spaceships: they can travel also 2 planets at a time.

  Example: *from Earth to Venus, Mars, Mercury or Jupiter.*

Spaceship can ONLY travel to a planet when it has free landing platforms. If not, then a message is displayed that „Can't travel at the moment, no free landing platforms".

All spaceships implement interface ISpaceship:

```
interface ISpaceShip
{
    void Travel(int destinationPlanetId);
}
```

Spaceships can be created in the programs main method. When a spaceship is created, it is added to planet Earth.

Testing

Add nUnit and minimum 6 tests. It's best to test after adding a new step.

Example usage:

```
class Program
{
      static void Main(string[] args)
      {
         PlanetarySystem planetarySystem = new PlanetarySystem(); //Planetary system with
8 planets is created
          SpaceShip spaceShipJyri = new SpaceShip("Jüri", planetarySystem); //spaceship
is created to the planet system and added to Earth

          planetarySystem.planets[2].PrintInfo(); //Prints Earths info
          planetarySystem.planets[2].PrintSpaceshipInfo(); //Prints Earths info of
spaceships, should contain Jüri

          spaceShipJyri.Drive(5); //A notification that the ship cant drive this far
          spaceShipJyri.Drive(3); //Info about driving to planet 3

          planetarySystem.planets[2].PrintSpaceshipInfo(); //Prints Earths info of
spaceships, all landing platforms should be empty
          planetarySystem.planets[3].PrintSpaceshipInfo(); //Prints Mars's info of
spaceships, should contain Jüri

          planetarySystem.PrintAllInfo(); //Prints info about all the planets in the
planetary system
        }
    }
```

General tips:

- The order of the planets is always the same! Keep them in an array and access them by their indexes.
- There should be only one planetary system. If you need to access planet Venus then you know it is the second planet in the Planets array in the planetary system.
- Start by doing the system without spaceships, add the spaceships as a last element.
- Use Jupiter as the test planet for calculations/info method (there are the example values for it).
- When spaceships are travelling then keep track of their previous location to check if their destination is in the valid range (that they move either one or two planets at a time).
- Think when is better to use array and when list.
- Think of the constructors for planets and spaceships!
- Follow clean code principles, use good naming!
- Do not use public fields, use properties (reminder: https://www.dotnetperls.com/property)
- Spaceship can not be on multiple planets at the same time

Example structure:

```
namespace ClassroomExample
{
    /* Example structure. Classroom has students and students can have homework tasks.
       Homework tasks can be assigned to students in the classroom*/

    class Program
    {
        static void Main(string[] args)
        {
            ClassRoom  roomA = new ClassRoom(); //creates a classroom with 2 students in there
            Homework hw1 = new Homework("Do programming tasks");
            hw1.AddHomework(1, roomA); //add this homework to the student with id 1 in classroomA
        }
    }

    class ClassRoom
    {
        public Student[] students; //array of student objects

        public ClassRoom()
        {
            students = new Student[2]; //array of 2 students
            students[0] =new Student("Mary"); //add Mary as the first student
            students[1] =new Student("Jane"); //add Janes as the second student
        }
    }

    class Student
    {
        public string name; //name of the student
        public List<Homework> homeworkList; //list of grades for the student
        public Student(string studentName) //constructor
        {
            name = studentName;
            homeworkList = new List<Homework>();
        }
    }

    class Homework
    {
        string description;
        ClassRoom classroom; //classroom for this homework
        public Homework(string homeworkDesciription, ClassRoom room) //constructor, uses the classroom as
parameter
        {
            description = homeworkDesciription;
            classroom = room;
        }

        public void AddHomework(int studentId)
        {
            Student studentToAddHomeworkTo =room.students[studentId]; //get the student from classroom by
index
            studentToAddHomeworkTo.homeworkList.Add(this); //adding the current homework item
            Console.WriteLine("Added homework with description {0} to student {1}", this.description,
studentToAddHomeworkTo.name);
        }
    }
}
```