# Exercises: Unit Testing Classes

Tasks for exercise in class and for homework to the course  ["Programming Advanced for QA" @ SoftUni](https://softuni.bg/trainings/4257/programming-advanced-for-qa-november-2023).

Submit your solutions here: <https://judge.softuni.org/Contests/4492/Objects-and-Classes-Unit-Testing-Exercise>

# Unit Test: Student

Look at the **provided skeleton** and examine the Student.cs class that you will test:

A screenshot of a computer

Description automatically generated

The class has **properties** for **first** **name**, **last** **name**, **age,** and **hometown**:

A screen shot of a computer code

Description automatically generated

It also has a **method** that takes in a **string** **array** representing **students** in the form of:

"{first\_name} {last\_name} {age} {hometown}"

Also, a string representing which **town** the method should **filter** the students by and **return** them as a string:

A screen shot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

Then, look at the tests inside the StudentTests.cs class:

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

Notice the use of a **setup** **method**, so each test has a brand new **student instance** to use.

The first test if **partially** **finished** so you have a **reference**, the rest are **empty,** and your task is to finish them. The tests should run when you're finished:

A close up of text

Description automatically generated

# Unit Test: Song

The class Song.cs has **properties** for **list type**, **name**, and **time**:

A screenshot of a computer code

Description automatically generated

It also has a **method** that takes in a **string** **array** representing **songs** in the form of:

"{type} {name} {time}"

Also, a string representing which **list** (**type**) the method should **retrieve** and **return** each song in it as a string:

A screen shot of a computer code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

Now, look at the tests inside the SongTests.cs class:

A screenshot of a computer program

Description automatically generated

You are given a **setup** **method** again as well as one **partially** **finished** test, the rest are **empty,** and your task is to finish them. The tests should run when you're finished:

A screenshot of a computer

Description automatically generated

# Unit Test: Store

The **folder** **Store** contains **3 classes**, in which the **main** one is Shop.cs.

The other 2 classes are smaller classes **representing real life objects** only holding **properties**.

Box.cs:

A screenshot of a computer program

Description automatically generated

Item.cs:

A close-up of text

Description automatically generated

Shop.cs only has a **method** which takes in a **string** **array** representing **products** in the form of:

"{serial\_number} {name} {quantity} {price}"

For each **product** a new Item is created and placed in a new Box then the box is **added** to a **list**. Finally, the list of boxes is returned as a **string of information**:

A screenshot of a computer program

Description automatically generated

A computer screen shot of a code

Description automatically generatedNow, look at the tests inside the ShopTests.cs class:

A screenshot of a computer program

Description automatically generated

This time write the **setup** **method** on your own. You are given **one partial test**, the rest are **empty,** and your task is to finish them. The tests should run when you're finished:

A black text on a white background

Description automatically generated

# Unit Test: Vehicle

The **folder** **Vehicle** contains **4 classes**, in which the **main** one is Vehicles.cs.

The other 3 classes **represent real life objects** only holding **properties**:

Car.cs:

A screenshot of a computer code

Description automatically generated

Catalogue.cs:

A screenshot of a computer program

Description automatically generated

Truck.cs:

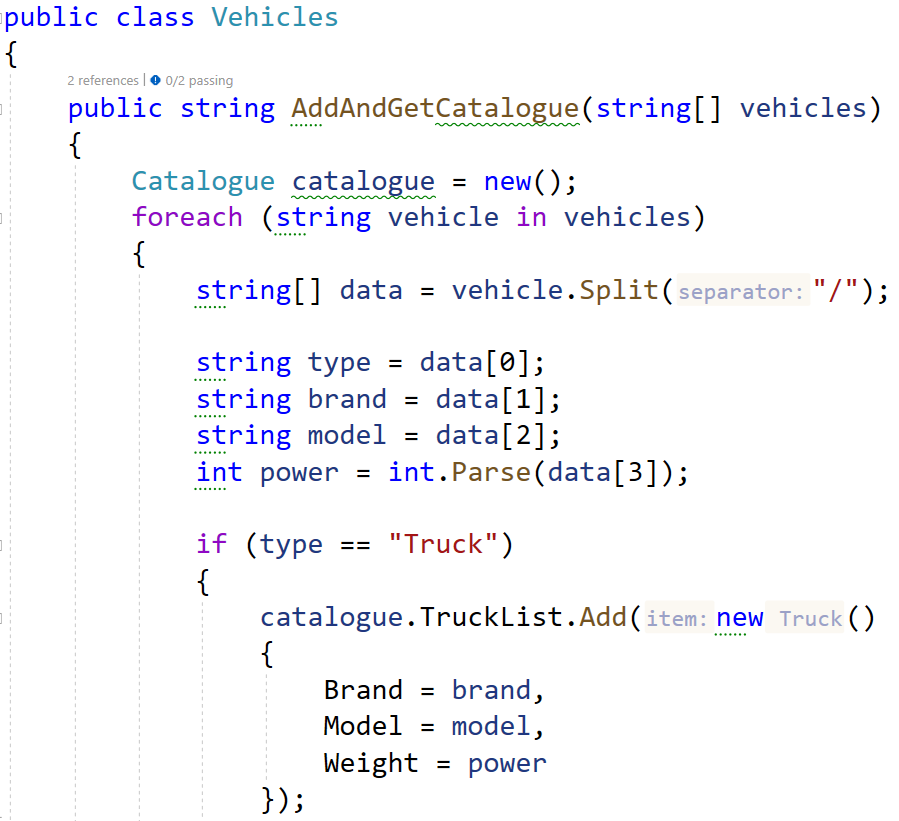
A screenshot of a computer code

Description automatically generated

Catalogue.cs only has a **method** which takes in a **string** **array** representing **vehicles** in the form of:

"{type}/{brand}/{model}/{power}"

First, a new Catalogue is created. For each **vehicle** a new Truck or Car based on the type is created and added to the relevant list in the catalogue. Finally, a string is returned based on the catalogue:



A computer code with text

Description automatically generated

A computer screen shot of text

Description automatically generated

Now, look at the tests inside the VehicleTests.cs class:

A screen shot of a computer program

Description automatically generated

Write the **setup** **method** on your own. You are given **one partial test**, the rest are **empty,** and your task is to finish them. The tests should run when you're finished:

A close up of a text

Description automatically generated

# Unit Test: Person

The class Person.cs has **properties** for **name**, **id**, and **age**:

A screenshot of a computer code

Description automatically generated

The first **method** it has, AddPeople(), takes in a **string** **array** representing **people** in the form of:

"{name} {id} {age}"

The method adds all people to a **list** and returns it:

A computer code with text

Description automatically generated with medium confidence

A computer screen shot of a computer code

Description automatically generated

The next method, GetByAgeAscending(), takes in a **list of peopl**e, **sorts the list by age**, and returns a **string** with **information**:

A computer code with many text

Description automatically generated with medium confidence

Now, look at the tests inside the PersonTests.cs class:

A screenshot of a computer program

Description automatically generated

Write the **setup** **method** on your own. You are given **one partial test**, the rest are **empty,** and your task is to finish them. The tests should run when you're finished:

A close up of a text

Description automatically generated

# Unit Test: Article

The class Article.cs has **properties** for **title**, **content**, **author**, and **article list**:

A screenshot of a computer program

Description automatically generated

The first **method** it has, AddArticles(), takes in a **string** **array** representing **articles** in the form of:

"{title} {content} {author}"

The method adds all articles to a **list** and returns it:

A screenshot of a computer code

Description automatically generated

The next method, GetArticleList(), takes in an **instance of an article**, and a **print criteria**. Based on the criteria it **orders the list,** and returns a **string** with **information**:

A screen shot of a computer program

Description automatically generated

A computer code on a white background

Description automatically generated

Now, look at the tests inside the ArticleTests.cs class:

A screenshot of a computer program

Description automatically generated

Write the **setup** **method** on your own. You are given **one partial test**, the rest are **empty,** and your task is to finish them. The tests should run when you're finished:

A close up of words

Description automatically generated

# Unit Test: Planet

The class Planet.cs has **properties** for **name**, **diameter**, **sun distance**, and **moon number**. It also has a **private** **field** for **gravitation** and a **constructor**:

A computer screen shot of a code

Description automatically generated

The first **method** it has, CalculateGravity(), takes in a **number** representing **mass.** The method calculates the **planets gravity** with a calculation:

A close-up of a computer screen

Description automatically generated

The next method, GetPlanetInfo(), returns a **string with information about the planet**:

A computer screen shot of a program

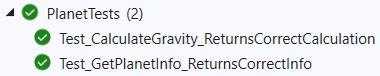
Description automatically generated

Now, look at the tests inside the PlanetTests.cs class:

A screen shot of a computer

Description automatically generated

You are given **one partial test**, the rest are **empty,** and your task is to finish them. The tests should run when you're finished:



At the end make sure all tests pass:

A screenshot of a computer

Description automatically generated