**GRID\_Problem 1 "Структура"**

**Due date**: Sunday, 7 October 2018, 16:00  
**Requested files**: StartUp.cs, Engine.cs, Tyre.cs, HardTyre.cs, UltrasoftTyre.cs, Driver.cs, AggressiveDriver.cs, EnduranceDriver.cs, Car.cs, RaceTower.cs ([Download](https://it-kariera.mon.bg/e-learning/mod/vpl/views/downloadrequiredfiles.php?id=1578))  
**Type of work**: Individual work  
**Настройки на оценките**: Максимална оценка: 30  
**Run**: Не. **Evaluate**: Да  
**Automatic grade**: Да.

**GRID**

**Преглед на задачата**

Трябва да напишете програма, която да симулира компютърната игра GRID под номера на команди. Различен брой шофьори може да участва във всяко състезание и всеки шофьор ... ами, кара кола. Шофьорите имат различно отношение към пистата и автомобилите имат различни спецификации, което прави състезанието вълнуващо!

**Настройка**

За да настроите проекта си, използвайте скелета, който ви е предоставен. Проектът трябва да има класовете "**StartUp**" и "**RaceTower**" с namespace "**GRID**". Спазването на това правило ще доведе до компилирането на кода ви. Може свободно да използвате папки с namespace имена за останалата част на проекта ви.

**Problem 1.       Структура**

**Driver**

Всички шофьори имат име, общо време и кола за шофиране:

Name – символен низ

TotalTime – реално число

Car – параметър от тип Car

**FuelConsumptionPerKm**– реално число

**Speed**– реално число

Скоростта на шофьора се изчислява по формулата по-долу. Имайте предвид, че скоростта се променя на всяка обиколка.

**Speed = “(car’s Hp + tyre’s degradation) / car’s fuelAmount”**

**AggressiveDriver**

Този тип водачи имат **FuelConsumptionPerKm** равен на **2.7 литра**. Също така **скоростта** му е умножена по **1.3**.

**EnduranceDriver**

и тип водачи имат **FuelConsumptionPerKm** равен на **1.5 литра**.

**Cars**

Всяка кола трябва да запази своята конски сили (Hp), количеството гориво и вида на гумите, които са в момента.

Hp –  цяло число

FuelAmount – реално число

Tyre – параметър от тип Tyre

**Максималният капацитет** на **горивния резервоар** за всеки автомобил е **160 литра**. Количеството гориво **не може да стане по-голямо от максималния капацитет на резервоара**. Ако ви се даде повече гориво, отколкото е необходимо, **трябва да напълните резервоара до максимум и нищо друго не се случва**.

Ако количеството гориво падне **под 0 литра, трябва да направите изключение и водачът не може да продължи състезанието**.

**Tyres**

Всеки тип гума има различна твърдост. Също така има ниво на разграждане, което е неговият живот:

Name – символине низ

Hardness –  реално число

Degradation**–**реално число

Всяка гума започва с деградация от 100 точки и пада надолу към 0. При всяка обиколка деградацията се намалява със стойността на твърдостта. Ако деградацията на гумата падне под 0 точки, гумата се взривява и водачът не може да продължи състезанието. Ако една гума се взриви, трябва да изхвърлите**излючение**.

**UltrasoftTyre**

Има допълнително свойство:

Grip –  положително реално число.

Името на гумата е "**Ultrasoft**".

При всяка обиколка деградацията пада с твърдостта си съчетана със сцеплението на гумата. Също така, този тип гума се взривява, когато разпадът на гумата падне под 30 точки.

**HardTyre**

Името на гумата е “**Hard**”.

**Problem 2.       Бизнес логика**

Всяко изпълнение на приложението симулира само едно състезание. В началото получавате информация за пистата (обиколки / дължина), след което **състезалите** са регистрирани. Началото на състезанието се отбелязва с първата команда CompleteLaps. Състезанието завършва, когато всички обиколки се извършват от шофьорите.

**The Controller Class**

Бизнес логиката на програмата трябва да бъде концентрирана около няколко команди. Внедрете клас, наречен **RaceTower**, който ще притежава главната функционалност, представена от тези публично методи:

|  |
| --- |
| RaceTower.cs |
| public void SetTrackInfo(int lapsNumber, int trackLength)  {      //TODO: Add some logic here …  }    public void RegisterDriver(List<string> commandArgs)  {      //TODO: Add some logic here …  }    public void DriverBoxes(List<string> commandArgs)  {      //TODO: Add some logic here …  }    public string CompleteLaps(List<string> commandArgs)  {      //TODO: Add some logic here …  }    public string GetLeaderboard()  {      //TODO: Add some logic here …  } |

**ЗАБЕЛЕЖКА: Не трябва да променяте нищо по методите. Трябва да имплементирате логиката на самите методи.**

Методът **SetTrackInfo**() трябва да инициализира общия брой и дължина на трасето.

**Команди**

Има няколко команди, които контролират бизнес логиката на приложението и трябва да го направите.

Те са посочени по-долу.

**RegisterDriver Команда**

Създава състезал и го регистрира в състезанието. Входните данни **може да не са винаги валидни**. Ако н**е можете** да създадете състезал с данните, предоставени по тази команда, **просто го пропуснете**. Всички успешно регистрирани състезатели трябва да бъдат запазени в клас **Racetower** във всякакъв тип структура от данни, осигурена от **.NET Framework (без персонализирани структури).**

**Параметри**

* type – символен низ равен на “Aggressive“ или “Endurance“
* name – символен низ
* hp – цяло число
* fuelAmount – реално число
* tyreType – **символен низ**
* tyreHardness – реално число

Ако типът на гумата е Ultrasoft, ще получите още един параметър:

* grip –  позитивно реално число

**Leaderboard Команда**

На първоя ред принтирайте:

Lap {currentLap}/{totalLaps}

На следващите редове всички състезателите трябва да се показват в реда на техния напредък в следния формат:

{position} {driverName} {totalTime/failureReason}

Състезателите трябва да бъдат подредени по общото време във възходящ ред. Всикчи състезатели с повреда трябва да бъдат принтирани след активните състезатели в реда на техния провал (последният неуспешен шофьор е на дъното).

**CompleteLaps Команда**

При тази команда всички водачи продължават състезанието със зададения брой обиколки. На всяка обиколка TotalTime на всеки водач трябва да се увеличи с резултата от следната формула:

**“60 / (дължината на трасето / скоростта на състезателя)”**

**След всяка обиколка трябва да изпълните действията по-долу в същата последователност:**

1. Намалете количеството гориво на всеки състезател по следната формула: **“trackLength \* driver’s fuelConsumptionPerKm”**.
2. Деградирайте гумата според вида й.

Ако имате по-голям брой обиколки от броя на обиколките, останали в състезанието, трябва да изхвърлите изключение с подходящото съобщение и да не увеличавате броя завършени обиколки.

**Параметри**

* numberOfLaps – цяло число

**Box Команда**

Makes a driver to box at the current lap which **adds 20 seconds** to his TotalTime and either **changes** his **tyres** with **new ones** of the specified type **or refills** with **fuel**.

Кара състезателя да посети бокса при текущата обиколка, което добавя 20 секунди към неговото време. При посещение на бокса състезателя може да смени гумите си с нови или да зареди с гориво.

**Параметри**

* reasonToBox – симвоилен низ равен на ChangeTyres или Refuel
* driversName – символине низ
* tyreType / fuelAmount – символиен низ, определящ типа на новите гуми / реално число определяюо количеството гориво за презареждане

Ако причината е смяна на гуми, ще получите допълнителни параметри:

* tyreHardness - реално число

Ако типът на гумата е Ultrasoft, ще получите 1 допълнителен параметър:

* grip –  позитивно реално число

**Допълнително действие**

**DNF**

Ако състезател спре поради някаква поврдеа, той вече не напредва в състезанието под командата CompleteLaps. Състезателите, които вече не се състезават, все още участват в дъното на класацията, в реда на техния провал (последният неуспешен шофьор е на дъното).

* Съобщението в случай на спукана гума трябва да бъде - "Blown Tyre"
* Съобщението в случай на липса на гориво трябва да бъде - "Out of fuel"
* На първия ред ще получите цяло число, представляващо броя на обиколките в състезанието
* На втория ред ще получите цяло число, представляващо дължината на пистата
* На следващите редове ще получите различни команди. Трябва да спрете да четете входа, когато шофьорите завършат всички обиколки в състезанието

**Problem 3.       Вход / Изход**

**Вход**

По-долу можете да видите формата, в който всяка команда ще бъде дадена във входа:

* RegisterDriver {type} {name} {hp} {fuelAmount} {tyreType} {tyreHardness}
* RegisterDriver {type} {name} {hp} {fuelAmount} Ultrasoft {tyreHardness} {grip}
* Leaderboard
* CompleteLaps {numberOfLaps}
* Box Refuel {driversName} {fuelAmount}
* Box ChangeTyres {driversName} Hard {tyreHardness}
* Box ChangeTyres {driversName} Ultrasoft {tyreHardness} {grip}

**Изход**

По-долу можете да видите кой изход трябва да бъде предоставен от командите.

**Leaderboard Команда**

Lap {current lap}/{total laps number}

На следващите редове всички състезатели трябва да се показват в реда на техния напредък в следния формат:

{position} {driverName} {totalTime / failureReason}

**CompleteLaps Команда**

Ако на конзолата получите невалиден брой обиколки:

“There is no time! On lap {current lap}.”

**Finish**

След завършването на всички обиколки, трябва да отпечатате победителя в конзолата в следния формат:

“{Driver’s name} wins the race for {TotalTime} seconds.”

TotalTime трябва да се закръгли до три цифри след десетичната запетая.

**Ограничения**

* Името на състезателя ще бъде символен низ, който може да съдържа всеки ASCII символ, с изключение на интервал.
* Имената на състезателите винаги ще бъдат уникални.
* Всички шофьори ще бъдат регистрирани преди началото на състезанието (няма да има команда RegisterDriver след първата команда CompleteLaps).
* Състезателите няма да бъдат пращани повече от веднъж в бокса на обиколка.
* Всяко състезание ще има победител.
* Няма да има невалидни входни данни.

**Примери**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 32  3  RegisterDriver Aggressive FirstDriver 650 140 Ultrasoft 0.2 3.8  RegisterDriver Endurance SecondDriver 467 78.48 Hard 0.8  RegisterDriver Endurance ThirdDriver 160 78.48 Ultrasoft 0.4 2.7  CompleteLaps 17  Leaderboard  Box Refuel SecondDriver 98.28  CompleteLaps 15 | Lap 17/32  1 ThirdDriver 2896.110  2 FirstDriver 6918.938  3 SecondDriver 7209.032  SecondDriver wins the race for 9838.183 seconds. |
| 10  5  RegisterDriver Aggressive FirstDriver 650 140 Ultrasoft 10.2 3.0  RegisterDriver Aggressive SecondDriver 650 140 Hard 3.9  RegisterDriver Endurance ThirdDriver 360 78.48 Ultrasoft 2.4 0.7  CompleteLaps 14  CompleteLaps 8  Leaderboard  Box ChangeTyres ThirdDriver Hard 0.3  CompleteLaps 2 | There is no time! On lap 0.  Lap 8/10  1 ThirdDriver 931.587  2 SecondDriver 1124.098  3 FirstDriver Blown Tyre  ThirdDriver wins the race for 1752.693 seconds. |
| 14  5  RegisterDriver Endurance FirstDriver 650 140 Hard 0.2  RegisterDriver Endurance SecondDriver 650 140 Ultrasoft 0.2 0.3  RegisterDriver Aggressive ThirdDriver 350 100 Ultrasoft 0.2 0.3  RegisterDriver Aggressive FourthDriver 450 60 Hard 1.2  CompleteLaps 1  Leaderboard  Box Refuel FourthDriver 168  CompleteLaps 6  Box Refuel FourthDriver 2  CompleteLaps 6  Leaderboard  CompleteLaps 1 | Lap 1/14  1 FirstDriver 64.286  2 SecondDriver 64.286  3 ThirdDriver 70.200  4 FourthDriver 143.000  Lap 13/14  1 SecondDriver 1353.124  2 FirstDriver 1357.036  3 FourthDriver 2122.949  4 ThirdDriver Out of fuel  SecondDriver wins the race for 1563.054 seconds. |

**Problem 4.       Точкуване**

Всяка различна задача ви дава точки:

* Структура - 30 точки
* Бизнес логика - 50 точки
* Вход/Изход - 20 точки

**Requested files**

**StartUp.cs**



1

2

3

4

5

6

7

8

9

10

namespace GRID

{

public class StartUp

{

public static void Main(string[] args)

{

}

}

}

**Engine.cs**

**Tyre.cs**

**HardTyre.cs**

**UltrasoftTyre.cs**

**Driver.cs**

**AggressiveDriver.cs**

**EnduranceDriver.cs**

**Car.cs**

**RaceTower.cs**

namespace GRID

{

using System;

using System.Collections.Generic;

public class RaceTower

{

public void SetTrackInfo(int lapsNumber, int trackLength)

{

//TODO: Add some logic here …

throw new NotImplementedException();

}

public void RegisterDriver(List<string> commandArgs)

{

//TODO: Add some logic here …

throw new NotImplementedException();

}

public void DriverBoxes(List<string> commandArgs)

{

//TODO: Add some logic here …

throw new NotImplementedException();

}

public string CompleteLaps(List<string> commandArgs)

{

//TODO: Add some logic here …

throw new NotImplementedException();

}

public string GetLeaderboard()

{

//TODO: Add some logic here …

throw new NotImplementedException();

}

}

}

**Execution files**

# -- nunit-console path\_to\_csproj --

#load common script and check programs

. common\_script.sh

check\_program mcs

check\_program mono

get\_source\_files cs

#compile

export MONO\_ENV\_OPTIONS=--gc=sgen

export MONO\_GC\_PARAMS=max-heap-size=64m

#mcs -target:library -r:NUnit.2.6.4/lib/nunit.framework.dll -pkg:dotnet -out:output.dll $SOURCE\_FILES

mcs -target:library -r:/usr/lib/cli/nunit.framework-2.6.3/nunit.framework.dll -pkg:dotnet -out:output.dll $SOURCE\_FILES

if [ -f output.dll ] ; then

cat common\_script.sh > vpl\_execution

echo "export MONO\_ENV\_OPTIONS=--gc=sgen" >> vpl\_execution

echo "export MONO\_GC\_PARAMS=max-heap-size=64m" >> vpl\_execution

#echo "mono ./NUnit.Runners.2.6.4/tools/nunit-console.exe output.dll -noresult" >> vpl\_execution

echo "mono /usr/lib/nunit/nunit-console.exe output.dll -noresult" >> vpl\_execution

chmod +x vpl\_execution

for FILENAME in $SOURCE\_FILES

do

grep -E "System\.Windows\.Forms" $FILENAME 2>&1 >/dev/null

if [ "$?" -eq "0" ] ; then

mv vpl\_execution vpl\_wexecution

break

fi

done

fi

**vpl\_evaluate.sh**

# -- nunit-console path\_to\_csproj --

#load common script and check programs

. common\_script.sh

check\_program mcs

check\_program mono

get\_source\_files cs

#compile

export MONO\_ENV\_OPTIONS=--gc=sgen

export MONO\_GC\_PARAMS=max-heap-size=64m

mcs -r:/usr/lib/cli/nunit.framework-2.6.3/nunit.framework.dll -pkg:dotnet -out:output.exe /main:UnitTests.MainTests $SOURCE\_FILES

if [ -f output.exe ] ; then

cat common\_script.sh > vpl\_execution

echo "export MONO\_ENV\_OPTIONS=--gc=sgen" >> vpl\_execution

echo "export MONO\_GC\_PARAMS=max-heap-size=64m" >> vpl\_execution

echo "mono output.exe -noresult" >> vpl\_execution

chmod +x vpl\_execution

for FILENAME in $SOURCE\_FILES

do

grep -E "System\.Windows\.Forms" $FILENAME 2>&1 >/dev/null

if [ "$?" -eq "0" ] ; then

mv vpl\_execution vpl\_wexecution

break

fi

done

fi

**MainTests.cs**

using System;

using System.Text;

namespace UnitTests

{

/// <summary>

/// Description of MainTests.

/// </summary>

public class MainTests

{

public static void Main(String[] args)

{

var tests = new StructureTests\_Grid();

int grade = 0;

int points= 3;

/\*... T001Test ...\*/

try {

tests.AssertExistingClasses();

Console.WriteLine(FormatOutput("Trial test", "No", null));

grade = 0;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Trial test", "No", e));

}

/\*... T01Test ...\*/

try {

tests.TestDriversConstructorsParameters\_AreInCorrectOrderAndType();

Console.WriteLine(FormatOutput("Racing test: StructureTest01", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest01", points.ToString(), e));

}

/\*... T02Test ...\*/

try {

tests.TestUltrasoftTyresConstructorsParameters\_AreInCorrectOrderAndType();

Console.WriteLine(FormatOutput("Racing test: StructureTest02", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest02", points.ToString(), e));

}

/\*... T03Test ...\*/

try {

tests.TestHardTyresConstructorsParameters\_AreInCorrectOrderAndType();

Console.WriteLine(FormatOutput("Racing test: StructureTest03", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest03", points.ToString(), e));

}

/\*... T04Test ...\*/

try {

tests.TestMethod04();

Console.WriteLine(FormatOutput("Racing test: StructureTest04", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest04", points.ToString(), e));

}

/\*... T05Test ...\*/

try {

tests.TestMethod05();

Console.WriteLine(FormatOutput("Racing test: StructureTest05", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest05", points.ToString(), e));

}

/\*... T06Test ...\*/

try {

tests.TestMethod06();

Console.WriteLine(FormatOutput("Racing test: StructureTest06", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest06", points.ToString(), e));

}

/\*... T07Test ...\*/

try {

tests.TestMethod07();

Console.WriteLine(FormatOutput("Racing test: StructureTest07", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest07", points.ToString(), e));

}

/\*... T08Test ...\*/

try {

tests.TestHardTyresConstructorsParameters\_AreInCorrectOrderAndType08();

Console.WriteLine(FormatOutput("Racing test: StructureTest08", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest08", points.ToString(), e));

}

/\*... T09Test ...\*/

try {

tests.AssertExistingClasses09();

Console.WriteLine(FormatOutput("Racing test: StructureTest09", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest09", points.ToString(), e));

}

/\*... T10Test ...\*/

try {

tests.TestMethod10();

Console.WriteLine(FormatOutput("Racing test: StructureTest10", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: StructureTest10", points.ToString(), e));

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Console.WriteLine(String.Format("Grade :=>> {0} marks\n",grade.ToString()));

}

private static String FormatOutput(String testName, String value, Exception e) {

var sb = new StringBuilder();

String grade = (e == null ? value : "0");

sb.Append(String.Format("Comment :=>> {0}: {1}. {2} marks\n", testName, (e == null ? "Passed" : "Failed"), grade));

if (e != null) {

sb.Append(String.Format("<|-- \n{0}\n --|>\n", e.Message));

}

return sb.ToString();

}

}

}

**StuctureTests\_Grid.cs**

using System;

using System.Linq;

using System.Reflection;

using NUnit.Framework;

using GRID;

[TestFixture]

public class StructureTests\_Grid

{

/\*... T001Test ...\*/

private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

private const string ClassNotExists = "Class '%s' does not exist";

// Class names

private string[] classes001 =

{

"AggressiveDriver",

"EnduranceDriver",

"Driver",

"HardTyre",

"UltrasoftTyre",

"Tyre",

"Car"

};

[Test]

public void AssertExistingClasses()

{

foreach (var className in classes001)

{

AssertClassExists(GetType(className));

}

}

private void AssertClassExists(Type className)

{

Assert.IsNotNull(className, ClassNotExists);

}

private static Type GetType(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

/\*... T01Test ...\*/

private string[] classes01 =

{

"AggressiveDriver",

"EnduranceDriver"

};

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

private Type[] ctorParameters01 = { typeof(string), GetType01("Car") };

[Test]

public void TestDriversConstructorsParameters\_AreInCorrectOrderAndType()

{

foreach (var className in classes01)

{

AssertConstructors01(GetType01(className));

}

}

private void AssertConstructors01(Type className)

{

ConstructorInfo[] constructors = className.GetConstructors(BindingFlags.Instance | BindingFlags.Public);

foreach (var ctor in constructors)

{

var parameters = ctor.GetParameters();

Assert.IsTrue(parameters.Length == ctorParameters01.Length);

for (int i = 0; i < ctorParameters01.Length; i++)

{

var test = parameters[i].ParameterType;

Assert.AreEqual(parameters[i].ParameterType, ctorParameters01[i], "Constructor parameters are in wrong order");

}

}

}

private static Type GetType01(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

/\*... T02Test ...\*/

private string[] classes02 =

{

"UltrasoftTyre"

};

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

private Type[] ctorParameters02 = { typeof(double), typeof(double) };

[Test]

public void TestUltrasoftTyresConstructorsParameters\_AreInCorrectOrderAndType()

{

foreach (var className in classes02)

{

AssertConstructors02(GetType02(className));

}

}

private static Type GetType02(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

private void AssertConstructors02(Type className)

{

ConstructorInfo[] constructors = className.GetConstructors(BindingFlags.Instance | BindingFlags.Public);

foreach (var ctor in constructors)

{

var parameters = ctor.GetParameters();

Assert.IsTrue(parameters.Length == ctorParameters02.Length);

for (int i = 0; i < ctorParameters02.Length; i++)

{

var test = parameters[i].ParameterType;

Assert.AreEqual(parameters[i].ParameterType, ctorParameters02[i], "Constructor parameters are in wrong order");

}

}

}

/\*... T03Test ...\*/

private string[] classes03 =

{

"HardTyre"

};

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

private Type[] ctorParameters03 = { typeof(double) };

[Test]

public void TestHardTyresConstructorsParameters\_AreInCorrectOrderAndType()

{

foreach (var className in classes03)

{

AssertConstructors03(GetType03(className));

}

}

private void AssertConstructors03(Type className)

{

ConstructorInfo[] constructors = className.GetConstructors(BindingFlags.Instance | BindingFlags.Public);

foreach (var ctor in constructors)

{

var parameters = ctor.GetParameters();

Assert.IsTrue(parameters.Length == ctorParameters03.Length);

for (int i = 0; i < ctorParameters03.Length; i++)

{

var test = parameters[i].ParameterType;

Assert.AreEqual(parameters[i].ParameterType, ctorParameters03[i], "Constructor parameters are in wrong order");

}

}

}

private static Type GetType03(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

/\*... T04Test ...\*/

// Class names

private string[] classes04 =

{

"AggressiveDriver",

"EnduranceDriver",

"Driver",

"HardTyre",

"UltrasoftTyre",

"Tyre",

"Car"

};

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

[Test]

public void TestMethod04()

{

foreach (var className in classes04)

{

AssertFields(GetType04(className));

}

}

private void AssertFields(Type className)

{

var fields = className.GetFields(BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public);

foreach (var fieldInfo in fields)

{

Assert.IsTrue(fieldInfo.IsPrivate, $"{fieldInfo.Name} in {className.Name} is NOT Private");

}

}

private static Type GetType04(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

/\*... T05Test ...\*/

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

// Class names

private Type[] classes05 =

{

GetType05("Driver"),

GetType05("Tyre")

};

[Test]

public void TestMethod05()

{

foreach (var className in classes05)

{

AssertProtectedCtors(className);

}

}

private void AssertProtectedCtors(Type className)

{

ConstructorInfo[] publicConstructors = className.GetConstructors(BindingFlags.Instance | BindingFlags.Public);

if (publicConstructors.Length > 0)

{

Assert.AreEqual(0, publicConstructors.Length, $"{className.Name} has public constructor!");

}

}

private static Type GetType05(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

/\*... T06Test ...\*/

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

// Class names

private Type[] classes06 =

{

GetType06("AggressiveDriver"),

GetType06("EnduranceDriver")

};

private Type parentClass = GetType06("Driver");

[Test]

public void TestMethod06()

{

foreach (var className in classes06)

{

Assert.IsTrue(CheckHierarchy06(className));

}

}

private bool CheckHierarchy06(Type className)

{

var baseClass = className.BaseType;

while (baseClass != typeof(object))

{

if (baseClass == parentClass)

{

return true;

}

baseClass = baseClass.BaseType;

}

return false;

}

private static Type GetType06(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

/\*... T07Test ...\*/

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

// Class names

private Type[] classes07 =

{

GetType07("UltrasoftTyre"),

GetType07("HardTyre")

};

private Type parentClass07 = GetType07("Tyre");

[Test]

public void TestMethod07()

{

foreach (var className in classes07)

{

Assert.IsTrue(CheckHierarchy07(className));

}

}

private bool CheckHierarchy07(Type className)

{

var baseClass = className.BaseType;

while (baseClass != typeof(object))

{

if (baseClass == parentClass07)

{

return true;

}

baseClass = baseClass.BaseType;

}

return false;

}

private static Type GetType07(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

/\*... T08Test ...\*/

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

private Type[] classes08 =

{

GetType08("Car")

};

private Type[] ctorParameters08 = { typeof(int), typeof(double), GetType08("Tyre") };

[Test]

public void TestHardTyresConstructorsParameters\_AreInCorrectOrderAndType08()

{

foreach (var className in classes08)

{

AssertConstructors08(className);

}

}

private void AssertConstructors08(Type className)

{

ConstructorInfo[] constructors = className.GetConstructors(BindingFlags.Instance | BindingFlags.Public);

foreach (var ctor in constructors)

{

var parameters = ctor.GetParameters();

Assert.IsTrue(parameters.Length == ctorParameters08.Length);

for (int i = 0; i < ctorParameters08.Length; i++)

{

var test = parameters[i].ParameterType;

Assert.AreEqual(parameters[i].ParameterType, ctorParameters08[i], "Constructor parameters are in wrong order");

}

}

}

private static Type GetType08(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

/\*... T09Test ...\*/

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

//private const string ClassNotExists = "Class '%s' does not exist";

// Class names

private string[] classes09 =

{

"AggressiveDriver",

"EnduranceDriver",

"Driver",

"HardTyre",

"UltrasoftTyre",

"Tyre",

"Car"

};

[Test]

public void AssertExistingClasses09()

{

foreach (var className in classes09)

{

AssertClassExists09(GetType09(className));

}

}

private void AssertClassExists09(Type className)

{

Assert.IsNotNull(className, ClassNotExists);

}

private static Type GetType09(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

/\*... T10Test ...\*/

//private static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

private Type sut = GetType10("Car");

[Test]

public void TestMethod10()

{

PropertyInfo tyreProperty = sut.GetProperty("Tyre");

PropertyInfo fuelAmountProperty = sut.GetProperty("FuelAmount", BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public);

MethodInfo tyreSetMethod = tyreProperty.SetMethod;

MethodInfo fuelAmountSetMethod = fuelAmountProperty.SetMethod;

Assert.IsTrue(!tyreSetMethod.IsPublic);

Assert.IsTrue(!fuelAmountSetMethod.IsPublic);

}

private static Type GetType10(string name)

{

var type = ProjectAssembly

.GetTypes()

.FirstOrDefault(t => t.Name == name);

return type;

}

}

**DriverFailiures.cs**

public class DriverFailiures

{

public const string BlownTyre = "Blown Tyre";

public const string OutOfFuel = "Out of fuel";

public const string Crashed = "Crashed";

}

**OutputMessages.cs**

public class OutputMessages

{

public const string InvalidNumberOfLaps = "There is no time! On lap {0}."; //Tested

public const string LapsCompleted = "Lap {0}/{1}"; //Tested

public const string Winner = "{0} wins the race for {1:f3} seconds.";

public const string SuccessfulOvertake = "{0} has overtaken {1} on lap {2}."; //Tested

}