**MiningSystem\_Тестове "Структура"**

**Avaliable from**: Sunday, 14 October 2018, 10:00  
**Due date**: Sunday, 14 October 2018, 16:00  
**Requested files**: StartUp.cs, Miner.cs, Driller.cs, Hewer.cs, Provider.cs, SunProvider.cs, ElectricityProvider.cs, SystemManager.cs ([Download](https://it-kariera.mon.bg/e-learning/mod/vpl/views/downloadrequiredfiles.php?id=1588))  
**Type of work**: Individual work  
**Настройки на оценките**: Максимална оценка: 30  
**Run**: Не. **Evaluate**: Да  
**Automatic grade**: Да.

**Mining System**

**Преглед на задачата**

Системата се състои от миньори и източници на енергия. Миньорите събират въглища, но имат нужда от много енергия, за да направят това. Тук се появяват източниците на енергия. Източниците на енергия предоставят енергия, която миньорите използват, за да събират въглища.

**Скелет**

В скелета ви е предоставена обработката на входа и празни методи в **SystemManager** **класа**. Допишете липсващата логика в класовете и в контролера, за да може приложението да работи по коректен начин.

**Task 1: Структура**

Структурата се състои от миньори и източници на енергия.

**Miners**

**Miner**

Всеки миньор има следните полета:

* id – низ
* coalOutput – реално число
* energyRequirement – реално число

За всеки миньор трябва да валидирате, че изходното количество въглища и необходимата енергия не е негативна. Също така трябва да валидирате, че необходимата енергия е не повече от **20000**. Има два вида миньори:

**Hewer**

Има допълнително поле:

* enduranceFactor – цяло число.

ПРИ ИНИЦИАЛИЗИРАНЕ разделете дадената необходима енергия на фактора за издръжливост.

**Driller**

ПРИ ИНИЦИАЛИЗИРАНЕувеличете изходното количество въглеща с 200% и необходимата енергия със 100%.

**Providers**

**Provider**

Всеки източник на енергия има следните полета:

* id – низ
* energyOutput – реално число

Изходната енергия трябва да бъде позитивно число, по-малко от **10000**.

**SunProvider**

При инициализация увеличете изходната енергия с 25%.

**ElectricityProvider**

При инициализация увеличете изходната енергия с 50%.

**Task 2: Бизнес логика**

**Клас Контролер**

Бизнес логиката на програмата трябва да бъде концентрирана около няколко команди. Имплементирайте клас, наречен **SystemManager**, който ще притежава основната функционалност, представена от тези публични методи:

|  |
| --- |
| SystemManager.cs |
| public string RegisterHarvester(List<string> arguments)  {      //TODO: Add some logic here …  }  public string RegisterProvider(List<string> arguments)  {      //TODO: Add some logic here …  }  public string Day()  {      //TODO: Add some logic here …  }  public string Check(List<string> arguments)  {      //TODO: Add some logic here …  }  public string ShutDown()  {      //TODO: Add some logic here …  } |

**ЗАБЕЛЕЖКА: Методите се извикват отвън, така че тези методи НЕ трябва да приемат командния параметър като част от аргументите**

**Функционалност**

Миньорите и източниците на енергия имат уникални идентификационни номера.

Когато отмине ден, източниците на енергия произвеждат енергия, а миньорите консумират енергия и събират въглища. В програмта един ден отминава при получена съответставща команда.

Източниците на енергия произвеждат енергия, която се съхранява в системата. Когато има достатъчно енергия за захранване на всички миньори, миньорите я консумират и връщат добитите въглищата.Системата пази **totalStoredEnergy** и **totalMinedCoal**.

**Команди**

Има няколко команди, които контролират бизнес логиката на приложението, което трябва да изградите.

Те са посочени по-долу.

**RegisterMiner**

Създава се миньор и го регистрира в системата.

**Парамертри**

* type – низ, равен на Driller или Hewer.
* id – низ
* coalOutput – позитивно реално число
* energyRequirement – позитивно реално число

Ако типът е Hewer, ще получите един допълнителен парамертър:

* enduranceFactor – позитивно цяло число.

**RegisterProvider**

Създава се източник на енергия и го регистрира в системата.

**Параметри**

* type – низ, който е равен на Sun или Electricity
* id – низ
* energyOutput – позитивно реално число

**Day**

Когато получите тази команда, един ден минава. Това е моментът, в който започва истинската работа. Трябва да изчислите цялата проивзедена енергия от източниците на енергия и да я съхраните в системата. След това трябва да проверите дали има достатъчно енергия в системата, за да започнат миньорите да копаят. Ако сумата от енергийните нужди на всички миньори е по-голяма от съхраняваната енергия, тогава нищо НЕ се случва. Ако има достатъчно енергия, ВСИЧКИ миньори започват да копаят и консумират от съхренаната енергия.

**Check**

Системата трябва да провери дали има елемент с дадения идентификатор между доставчиците на енергия или миньорите, връщайки стринговата репрезентация на обекта. Идентификационните номера са уникални, така че трябва да има само едно с този номер.

**Параметри**

* id – низ

**Shutdown**

Спира програмата и отпечатва оснталата съхранена енергия и събраното количество въглища.

**Task 3: Вход / Изход**

**Вход**

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

* RegisterMiner Driller {id} {coalOutput} {energyRequirement}
* RegisterMiner Hewer {id} {coalOutput} {energyRequirement} {enduranceFactor}
* RegisterProvider {type} {id} {energyOutput}
* Day
* Check {id}
* Shutdown

**Изход**

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

**RegisterHarvester**

При успешно изпълнена команда трябва да отпечата "Successfully registered {type} Miner – {id}". В противен случай изпечатайте: "Miner is not registered, because of it's {propertyName}"

**RegisterProvider**

При успешно изпълнена команда трябва да отпечата "Successfully registered {type} Provider – {id}”. В противен случай изпечатайте: "Provider is not registered, because of it's {propertyName}"

**Day**

Трябва да изведете следното съобщение:

“A day has passed.

“Energy Provided: {summedEnergyOutput}”.

“Mined Coal: {summedCoalOutput}”.

summedEnergyOutput и summedCoalutput са количествата, които са били произведени за деня.

**Check**

Трябва да върне стринговата репрезентация на елемента с дадения идентификационен номер. Ако няма такъв елемент, командата трябва да върне следното съобщение: “No element found with id – {id}”.

Тъй като елементът може да бъде доставчик на енергия или миьнор, двата изходни формати са предоставени по-долу:

|  |  |
| --- | --- |
| Miner | Provider |
| “{type} Miner - {id}    Coal Output: {coalOutput}    Energy Requirement: {energyRequired}” | “{type} Provider - {id}    Energy Output: {energyOutput}” |

**Shutdown**

Трябва да изведете следното съобщение:

 “System Shutdown

 Total Energy Stored: {totalEnergyStored}

 Total Mined Coal: {totalMinedCoal}”.

totalEnergyStored and totalMinedCoal са общите стойности, които са събрани по време на изпълнението на програмата.

**Ограничения**

* Идентификационния номер ще бъде низ, който може да съдържа всеки символ от ASCII таблицата без празно пространство (' ').
* Идентификационните номера ще бъдат уникални.
* Реалните числа ще бъдат в диапазона**[-1.000.000, 1.000.000]**.
* enduranceFactor ще бъде цяло число в диапазона**[1, 10]**.

**Примери**

|  |  |
| --- | --- |
| **Input** | **Output** |
| RegisterMiner Hewer AS-51 100 100 10  RegisterMiner Driller CDD 100 50  RegisterProvider Sun Falcon 100  Day  Check AS-51  Check CDD  Check Falcon  Day  Shutdown | Successfully registered Hewer Miner - AS-51  Successfully registered Driller Miner - CDD  Successfully registered Sun Provider - Falcon  A day has passed.  Energy Provided: 125  Mined Coal: 400  Hewer Miner - AS-51  Coal Output: 100  Energy Requirement: 10  Driller Miner - CDD  Coal Output: 300  Energy Requirement: 100  Sun Provider - Falcon  Energy Output: 125  A day has passed.  Energy Provided: 125  Mined Coal: 400  System Shutdown  Total Energy Stored: 30  Total Mined Coal: 800 |
| RegisterMiner Hewer AS-51 100 1000000 10  RegisterMiner Driller CDD 100 50  RegisterProvider Sun Falcon 100  RegisterProvider Sun Pesho 100000  Day  Check CDD  Check Falcon  Day  Shutdown | Miner is not registered, because of it's EnergyRequirement  Successfully registered Driller Miner - CDD  Successfully registered Sun Provider - Falcon  Provider is not registered, because of it's EnergyOutput  A day has passed.  Energy Provided: 125  Mined Coal: 300  Driller Miner - CDD  Coal Output: 300  Energy Requirement: 100  Sun Provider - Falcon  Energy Output: 125  A day has passed.  Energy Provided: 125  Mined Coal: 300  System Shutdown  Total Energy Stored: 50  Total Mined Coal: 600 |
| RegisterProvider Electricity Deep-1 1000  RegisterProvider Electricity Deep-3 2000  Day  RegisterMiner Driller S-1 10000 11250  Day  Check Something  Check S-1  Day  Shutdown | Successfully registered Electricity Provider - Deep-1  Successfully registered Electricity Provider - Deep-3  A day has passed.  Energy Provided: 4500  Mined Coal: 0  Miner is not registered, because of it's EnergyRequirement  A day has passed.  Energy Provided: 4500  Mined Coal: 0  No element found with id - Something  No element found with id - S-1  A day has passed.  Energy Provided: 4500  Mined Coal: 0  System Shutdown  Total Energy Stored: 13500  Total Mined Coal: 0 |

**Requested files**

**StartUp.cs**

using System;

using System.Collections.Generic;

using System.Linq;

namespace MiningSystem

{

public class StartUp

{

private static bool isRunning = true;

private static void Main()

{

SystemManager systemManager = new SystemManager();

while (isRunning)

{

string input = Console.ReadLine();

List<string> parsedInput = ParseInput(input);

string commandResult = DispatchCommand(parsedInput, systemManager);

Console.WriteLine(commandResult);

}

}

private static string DispatchCommand(List<string> arguments, SystemManager manager)

{

string command = arguments[0];

arguments.RemoveAt(0);

string result = null;

switch (command)

{

case "RegisterMiner":

result = manager.RegisterMiner(arguments);

break;

case "RegisterProvider":

result = manager.RegisterProvider(arguments);

break;

case "Day":

result = manager.Day();

break;

case "Check":

result = manager.Check(arguments);

break;

case "Shutdown":

result = manager.ShutDown();

isRunning = false;

break;

}

return result;

}

private static List<string> ParseInput(string input)

{

return input.Split().ToList();

}

}

}

**Miner.cs**

**Driller.cs**

**Hewer.cs**

**Provider.cs**

**SunProvider.cs**

**ElectricityProvider.cs**

**SystemManager.cs**

using System;

using System.Collections.Generic;

namespace MiningSystem

{

public class SystemManager

{

public string RegisterMiner(List<string> arguments)

{

throw new NotImplementedException();

}

public string RegisterProvider(List<string> arguments)

{

throw new NotImplementedException();

}

public string Day()

{

throw new NotImplementedException();

}

public string Check(List<string> arguments)

{

throw new NotImplementedException();

}

public string ShutDown()

{

throw new NotImplementedException();

}

}

}

**Execution files**

**vpl\_run.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 -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\_MiningSystem();

int grade = 0;

int points= 3;

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

try {

tests.AssertExistingClasses();

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

grade = 0;

} catch (Exception e) {

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

}

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

try {

tests.TestMethod01();

Console.WriteLine(FormatOutput("Racing test: Test01", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test01", points.ToString(), e));

}

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

try {

tests.TestMethod02();

Console.WriteLine(FormatOutput("Racing test: Test02", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test02", points.ToString(), e));

}

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

try {

tests.TestMethod03();

Console.WriteLine(FormatOutput("Racing test: Test03", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test03", points.ToString(), e));

}

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

try {

tests.TestMethod04();

Console.WriteLine(FormatOutput("Racing test: Test04", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test04", points.ToString(), e));

}

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

try {

tests.TestMethod05();

Console.WriteLine(FormatOutput("Racing test: Test05", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test05", points.ToString(), e));

}

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

try {

tests.TestMethod06();

Console.WriteLine(FormatOutput("Racing test: Test06", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test06", points.ToString(), e));

}

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

try {

tests.TestMethod07();

Console.WriteLine(FormatOutput("Racing test: Test07", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test07", points.ToString(), e));

}

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

try {

tests.TestMethod08();

Console.WriteLine(FormatOutput("Racing test: Test08", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test08", points.ToString(), e));

}

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

try {

tests.TestMethod09();

Console.WriteLine(FormatOutput("Racing test: Test09", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test09", points.ToString(), e));

}

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

try {

tests.TestMethod10();

Console.WriteLine(FormatOutput("Racing test: Test10", points.ToString(), null));

grade += points;

} catch (Exception e) {

Console.WriteLine(FormatOutput("Racing test: Test10", 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 MiningSystem;

[TestFixture]

public class StructureTests\_MiningSystem

{

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

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

// Class names

private Type[] classes001 =

{

GetType("Miner"),

GetType("Hewer"),

GetType("Driller"),

GetType("Provider"),

GetType("SunProvider"),

GetType("ElectricityProvider")

};

[Test]

public void AssertExistingClasses()

{

foreach (var className in classes001)

{

AssertClassExists(className);

}

}

private void AssertClassExists(Type className)

{

Assert.IsNotNull(className, $"Class {className.Name} doesn't exists");

}

private static Type GetType(string name)

{

var type = ProjectAssembly

.GetTypes()

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

return type;

}

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

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

// Class names

private Type[] classes01 =

{

GetType01("ElectricityProvider"),

GetType01("SunProvider")

};

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

[Test]

public void TestMethod01()

{

foreach (var className in classes01)

{

AssertConstructors(className);

}

}

private void AssertConstructors(Type className)

{

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

foreach (var ctor in constructors)

{

var parameters = ctor.GetParameters();

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

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

{

var test = parameters[i].ParameterType;

Assert.AreEqual(parameters[i].ParameterType, ctorParameters[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 static readonly Assembly ProjectAssembly = typeof(StartUp).Assembly;

// Class names

private Type[] classes02 =

{

GetType02("Driller")

};

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

[Test]

public void TestMethod02()

{

foreach (var className in classes02)

{

AssertConstructors02(className);

}

}

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");

}

}

}

private static Type GetType02(string name)

{

var type = ProjectAssembly

.GetTypes()

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

return type;

}

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

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

// Class names

private Type[] classes03 =

{

GetType03("Hewer")

};

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

[Test]

public void TestMethod03()

{

foreach (var className in classes03)

{

AssertConstructors03(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 ...\*/

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

// Class names

private Type[] classes04 =

{

GetType04("Miner"),

GetType04("Driller"),

GetType04("Hewer"),

GetType04("Provider"),

GetType04("SunProvider"),

GetType04("ElectricityProvider")

};

[Test]

public void TestMethod04()

{

foreach (var className in classes04)

{

AssertFields(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("Miner"),

GetType05("Driller"),

GetType05("Hewer"),

GetType05("Provider"),

GetType05("SunProvider"),

GetType05("ElectricityProvider")

};

[Test]

public void TestMethod05()

{

foreach (var className in classes05)

{

AssertFields05(className);

}

}

private void AssertFields05(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 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("SunProvider"),

GetType06("ElectricityProvider")

};

private Type parentClass = GetType("Provider");

[Test]

public void TestMethod06()

{

foreach (var className in classes06)

{

Assert.IsTrue(CheckHierarchy(className));

}

}

private bool CheckHierarchy(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("Hewer"),

GetType07("Driller")

};

private Type parentClass07 = GetType("Miner");

[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 sut = GetType08("Miner");

[Test]

public void TestMethod08()

{

var propertyInfo = sut.GetProperty("EnergyRequirement");

var setMethod = propertyInfo.SetMethod;

Assert.IsTrue(!setMethod.IsPublic);

}

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 Type sut09 = GetType09("Miner");

[Test]

public void TestMethod09()

{

var propertyInfo = sut09.GetProperty("CoalOutput");

var setMethod = propertyInfo.SetMethod;

Assert.IsTrue(!setMethod.IsPublic);

}

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 sut10 = GetType10("Provider");

[Test]

public void TestMethod10()

{

var propertyInfo = sut10.GetProperty("EnergyOutput");

var setMethod = propertyInfo.SetMethod;

Assert.IsTrue(!setMethod.IsPublic);

}

private static Type GetType10(string name)

{

var type = ProjectAssembly

.GetTypes()

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

return type;

}

}