# C# OOP Retake Exam – April 2021

# Easter

## Overview

Easter is coming there are eggs to be colored. You can't manage on your own, so Bunnies are helping you. Your task is to create an **Easter** project, where different types of **Bunnies** color **Eggs**. Naturally, each Bunny has an energy level, which drops while working on an **Egg**, and they are running out of **Dyes**, again while working on an Egg.

## Setup

* Upload **only the** Easterproject in every problem **except** **Unit Tests**
* **Do not modify the interfaces or their namespaces**
* Use **strong cohesion** and **loose coupling**
* **Use inheritance and the provided interfaces wherever possible**.
  + This includes **constructors**, **method parameters** and **return types**
* **Do not** violate your **interface** **implementations** by adding **more public methods** or **properties** in the concrete class than the interface has defined
* Make sure you have **no public fields** anywhere

## Task 1: Structure (50 points)

For this task’s evaluation logic in the methods isn’t included.

You are given interfaces, and you have to implement their functionality in the **correct classes**.

There are **4** types of entities in the application: **Bunny, Egg, Workshop and Dye**. There should also be **BunnyRepository and EggRepository**.

### Bunny

Bunny is a **base class** of any **type of bunny** and it **should not be able to be instantiated**.

#### Data

* **Name** - **string**
  + If the name **is null or whitespace,** throw an **ArgumentException** with message:

**"Bunny name cannot be null or empty."**

* + All names will be unique
* **Energy** - **int**
  + The energy of a bunny
  + If a Bunny’s **energy drops below 0**, **set it to 0**
* Dyes - ICollection<I**Dye>**
  + A collection of a bunny's dyes

#### Constructor

A **Bunny** should take the following values upon initialization:

string name, int energy

#### Behavior

##### abstract void Work()

The **Work()** method decreases the bunny's energy by 10.

* If a Bunny’s energy **drops below 0**, **set it to 0**.

##### void AddDye(IDye Dye)

This method **adds** the given **Dye** to the Bunny's **collection** of Dyes.

#### Child Classes

There are several concrete types of **Bunny**:

##### HappyBunny

**Has 100 initial energy.**

A **HappyBunny** should take the following values upon initialization:

string name

##### SleepyBunny

**Has 50 initial energy.**

A **SleepyBunny** should take the following values upon initialization:

string name

###### **Behavior**

The method Work() **decreases** the Bunny's energy by additional **5 units** (15 in total).

### Dye

The **Dye** is a class that represents the tool, which a **Bunny** uses to color **Egg**.

#### Data

* **Power** - **int**
  + The power of an Dye
  + If the power is below **0,** **set it to 0**.

#### Constructor

An **Dye** should take the following values upon initialization:

int power

#### Behavior

##### void Use()

The **Use()** method decreases the Dye's power by 10.

* An Dye's power should **not** drop **below** **0**, if the power becomes less than 0, set it to 0

**bool IsFinished()**

* This method returns **true** if the **power** is **equal** to **0**

### Egg

This is the class which holds information about the **Egg** that a **Bunny** is working on.

#### Data

* **Name** - **string**
  + The name of a Egg
  + If the name **is null or whitespace,** throw an **ArgumentException** with message:

**"Egg name cannot be null or empty."**

* **EnergyRequired** - **int**
  + The energy an egg requires in order to be colored
  + If the **energyRequired** is **below 0**, **set it to 0**

#### Constructor

An **Egg** should take the following values upon initialization:

string name, int energyRequired

#### Behavior

##### void GetColored()

The **GetColored()** method **decreases** the required energy of the egg by **10 units**.

* An egg's required energy should **not** drop **below** **0**.

##### bool IsDone()

The **IsDone()** method returns **true** if the **energyRequired** is equal to **0**.

### Workshop

The **Workshop** class holds the main action, which is the **Color** method.

#### Constructor

A **Workshop** should take no values upon initialization.

#### Behavior

**void Color(IEgg Egg, IBunny Bunny)**

Here is how the **Color** method works:

* The bunny starts coloring the egg. This is only possible, if the bunny has energy and an dye that isn't finished.
* At the same time the egg is getting colored, so call the **GetColored()** method for the egg.
* Keep working **until** the egg is **done** or the bunny has **energy** and **dyes** to use.
* If at some point the **power** of the current dye **reaches** or **drops** **below 0**, meaning it is **finished**, then the Bunny should take the **next Dye** from its collection, if it has **any** **left**.

### BunnyRepository

The Bunny repository is a repository for the bunnies working for you.

#### Data

* Models - **a** **collection of bunnies (unmodifiable)**

#### Behavior

##### void Add(IBunny Bunny)

* **Adds** a **bunny** in the **collection**.
* Every bunny is **unique** and it is guaranteed that there will not be a Bunny with the same name

**bool Remove(IBunny Bunny)**

* **Removes** a **bunny** from the **collection**. **Returns true** if the deletion was **sucessful**, **otherwise** - **false**.

**IBunny FindByName(string name)**

* **Returns** the **first** **bunny** with the **given name**, if such exists. **Otherwise**, returns **null**.

### EggRepository

The Egg repository is a repository for eggs that await to be colored.

#### Data

* Models - **a** **collection of Eggs (unmodifiable)**

#### Behavior

##### void Add(IEgg Egg)

* **Adds** an **egg** in the **collection**.
* Every egg is **unique** and it is guaranteed that there will not be a egg with the same name

**bool Remove(IEgg Egg)**

* **Removes** a **egg** from the **collection**. **Returns true** if the deletion was **sucessful**, **otherwise** - **false**.

**IEgg FindByName(string name)**

* **Returns** the **first** **egg** with the **given name**, if such exists. **Otherwise**, returns **null**.

## Task 2: Business Logic (150 points)

### The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces, which you have to implement in the correct classes.

**Note: The** Controller **class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!**

The first interface is **I**Controller. You must create a Controllerclass, which implements the interface and implements all of its methods. The constructor of Controllerdoes not take any arguments. The given methods should have the logic described for each in the Commands section.

### Data

You need to keep track of some things, this is why you need some private fields in your controller class:

* **bunnies** - **BunnyRepository**
* **eggs** - **EggRepository**

### Commands

There are several **commands**, which control the **business** **logic** of the **application**. They are **stated** **below**.

#### AddBunny Command

##### Parameters

* bunnyType - string
* bunnyName - string

##### Functionality

**Adds** a bunny. **Valid** types are: "**HappyBunny**" and "**SleepyBunny**".

If the **bunny** **type** is **invalid**, you have to **throw an InvalidOperationException** with **the following message:**

* **"Invalid bunny type."**

If the **bunny** is **added successfully**, the method should **return** the following **string**:

* **"Successfully added {bunnyType} named {bunnyName}."**

**Note**: **Do not use Reflection** for the **method above**!

#### AddDyeToBunny Command

##### Parameters

* **bunnyName** - **string**
* **power - int**

##### Functionality

Creates a dye with the given power and adds it to the collection of the bunny.

If the bunny doesn't exist, throw an **InvalidOperationException** with message:

**"The bunny you want to add a dye to doesn't exist!"**

The method should **return** the following message:

**"Successfully added dye with power {dyePower} to bunny {bunnyName}!"**

#### AddEgg Command

##### Parameters

* **eggName** - **string**
* **energyRequired - int**

##### Functionality

Creates an **egg** with the provided **name** and **required energy**.

The method should **return** the following message:

**"Successfully added egg: {eggName}!"**

#### ColorEgg Command

##### Parameters

* eggName - string

##### Functionality

When the color command is called, the action happens.

You should start coloring the given egg, by assigning bunnies which are most ready (first the bunnies with the most energy):

* The bunnies that you should select are the ones with energy **equal to or above 50 units**.
* The **suitable ones** start working on the given egg.
* If a bunny’s **energy becomes 0**, **remove it from the repository**.
* If no **bunnies are ready**, throw **InvalidOperationException** with the following message:

**"There is no bunny ready to start coloring!"**

* After the work is done, you must return the following message, reporting whether the Egg is done:

**"Egg {eggName} is {done/not done}."**

**Note:** The **name** of the **egg** you receive will always be a **valid** one.

#### Report Command

##### Functionality

Returns information about **colored** **eggs** and **bunnies**:

**"****{countColoredEggs} eggs are done!"**

**"****Bunnies info:"**

**"****Name: {bunnyName1}"**

**"****Energy: {bunnyEnergy1}"**

**"****Dyes: {countDyes} not finished"**

**…**

**"Name: {bunnyNameN}"**

**"Energy: {bunnyEnergyN}"**

**"Dyes {countDyes} not finished left"**

**Note: Use \r\n or Environment.NewLine for a new line.**

#### Exit Command

##### Functionality

Ends the program.

### Input / Output

You are provided with one interface, which will help you with the correct execution process of your program. The interface is IEngine and the class implementing this interface should read the input and when the program finishes, this class should print the output.

#### Input

Below, you can see the **format** in which **each command** will be given in the input:

* **AddBunny** **{bunnyType} {bunnyName}**
* **AddEgg** **{eggName} {energyRequired}**
* **AddDyeToBunny** **{bunnyName} {power}**
* **ColorEgg {eggName}**
* **Report**
* **Exit**

#### Output

Print the output from each command when issued. If an exception is thrown during any of the commands' execution, print the exception message.

#### Examples

|  |
| --- |
| **Input** |
| **AddBunny SleepyBunny SleepyHead**  **AddBunny HappyBunny Sunshine**  **AddBunny InvalidBunny Sonny**  **AddDyeToBunny SleepyHead 10**  **AddDyeToBunny Sunshine 20**  **AddDyeToBunny Sunshine 20**  **AddDyeToBunny Sunshine 30**  **AddDyeToBunny Sunshine 10**  **AddDyeToBunny Sunshine 30**  **AddDyeToBunny Sunshine 20**  **AddDyeToBunny Sunshine 40**  **AddEgg Green 20**  **AddEgg Red 20**  **AddEgg Yellow 50**  **ColorEgg Green**  **ColorEgg Red**  **ColorEgg Yellow**  **Report**  **Exit** |
| **Output** |
| **Successfully added SleepyBunny named SleepyHead.**  **Successfully added HappyBunny named Sunshine.**  **Invalid bunny type**  **Successfully added dye with power 10 to bunny SleepyHead!**  **Successfully added dye with power 20 to bunny Sunshine!**  **Successfully added dye with power 20 to bunny Sunshine!**  **Successfully added dye with power 30 to bunny Sunshine!**  **Successfully added dye with power 10 to bunny Sunshine!**  **Successfully added dye with power 30 to bunny Sunshine!**  **Successfully added dye with power 20 to bunny Sunshine!**  **Successfully added dye with power 40 to bunny Sunshine!**  **Successfully added egg: Green!**  **Successfully added egg: Red!**  **Successfully added egg: Yellow!**  **Egg Green is done.**  **Egg Red is done.**  **Egg Yellow is done.**  **3 eggs are done!**  **Bunnys info:**  **Name: SleepyHead**  **Energy: 50**  **Dyes: 1 not finished**  **Name: Sunshine**  **Energy: 10**  **Dyes: 3 not finished** |

|  |
| --- |
| **Input** |
| **AddBunny SleepyBunny Moony**  **AddBunny SleepyBunny Latey**  **AddBunny HappyBunny Mikey**  **AddBunny HappyBunny Crispy**  **AddDyeToBunny Moony 20**  **AddDyeToBunny Mikey 180**  **AddDyeToBunny Moony 10**  **AddDyeToBunny Latey 10**  **AddDyeToBunny Crispy 20**  **AddDyeToBunny Crispy 10**  **AddDyeToBunny Crispy 10**  **AddEgg Spicy 100**  **AddEgg Beauty 160**  **AddEgg Purple 40**  **AddEgg PurpleRain 500**  **AddEgg Pink 300**  **ColorEgg Spicy**  **ColorEgg Beauty**  **ColorEgg Purple**  **ColorEgg PurpleRain**  **ColorEgg Pink**  **Report**  **Exit** |
| **Output** |
| **Successfully added SleepyBunny named Moony.**  **Successfully added SleepyBunny named Latey.**  **Successfully added HappyBunny named Mikey.**  **Successfully added HappyBunny named Crispy.**  **Successfully added dye with power 20 to bunny Moony!**  **Successfully added dye with power 180 to bunny Mikey!**  **Successfully added dye with power 10 to bunny Moony!**  **Successfully added dye with power 10 to bunny Latey!**  **Successfully added dye with power 20 to bunny Crispy!**  **Successfully added dye with power 10 to bunny Crispy!**  **Successfully added dye with power 10 to bunny Crispy!**  **Successfully added egg: Spicy!**  **Successfully added egg: Beauty!**  **Successfully added egg: Purple!**  **Successfully added egg: PurpleRain!**  **Successfully added egg: Pink!**  **Egg Spicy is done.**  **Egg Beauty is not done.**  **Egg Purple is not done.**  **Egg PurpleRain is not done.**  **Egg Pink is not done.**  **1 eggs are done!**  **Bunnys info:**  **Name: Moony**  **Energy: 5**  **Dyes: 0 not finished**  **Name: Latey**  **Energy: 35**  **Dyes: 0 not finished**  **Name: Crispy**  **Energy: 60**  **Dyes: 0 not finished** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with **Present** and **Bag** classes inside. The class will have some methods, fields and one constructor, which are working properly. You are **NOT ALLOWED** to change any class. Cover the whole class with unit tests to make sure that the class is working as intended.

You are provided with a **unit test project** in the **project skeleton**.

Do **NOT** use **Mocking** in your unit tests!

[TestFixture]

public class PresentsTests

{

private Bag bag;

[SetUp]

public void SetUp()

{

bag = new Bag();

}

[Test]

public void ConstructorInitializer()

{

Assert.That(bag,Is.Not.Null);

}

[Test]

public void CtorPresent()

{

Present present = new Present("name", 10);

Assert.AreEqual(present.Name, "name");

Assert.AreEqual(present.Magic, 10);

}

[Test]

public void CreateBagReturnsNullException()

{

Present present = null;

Assert.Throws<ArgumentNullException>(() => bag.Create(present));

}

[Test]

public void CreateReturnsOperationException()

{

Present present = new Present("name", 10);

bag.Create(present);

Assert.Throws<InvalidOperationException>(() => bag.Create(present));

}

[Test]

public void CreateReturnSuccessful()

{

Assert.DoesNotThrow(() => bag.Create(new Present("name", 10)));

}

[Test]

public void CreateIsSuccessful()

{

List<Present> presents = new List<Present>();

presents.Add(new Present("name", 10));

Assert.That(presents.Count, Is.EqualTo(1));

}

[Test]

public void RemoveIsSuccessful()

{

Present present = new Present("name", 10);

bag.Create(present);

Assert.DoesNotThrow(() => bag.Remove(present));

}

[Test]

public void GetPresentWithLeastMagicIsCorrect()

{

List<Present> presents = new List<Present>();

Present present = new Present("name", 10);

Present present2 = new Present("name2", 20);

bag.Create(present);

bag.Create(present2);

Assert.AreEqual(present, bag.GetPresentWithLeastMagic());

}

[Test]

public void GetPresentIsCorrect()

{

List<Present> presents = new List<Present>();

Present present = new Present("name", 10);

Present present2 = new Present("name2", 20);

bag.Create(present);

bag.Create(present2);

Assert.AreEqual(present, bag.GetPresent("name"));

}

[Test]

public void GetPresentsCollection()

{

Assert.That(bag.GetPresents(), Is.InstanceOf<IReadOnlyCollection<Present>>());

}