Circustrein – Unittests

Train

```
5 references
public class Train
{

    // Readonly so the object can only be set once  in constructor.
    private readonly List<Container> _containers;
    1 reference | 1/1 passing
    public IReadOnlyCollection<Container> Containers => _containers.AsReadOnly();

    2 references | 1/1 passing
    public Train()
    {
        _containers = new List<Container>();
    }

    6 references | 1/1 passing
    public void AddAnimalToTrain(Animal animal)
    {
        // If animal does not fit in any existing container, a new container is created.
        if (!TryToAddAnimalToAnyContainer(animal))
        {
            _containers.Add(item: new Container(animal));
        }
    }
    // Tries to add an animal to any container, when true returns true.
    1 reference
    private bool TryToAddAnimalToAnyContainer(Animal animal) => _containers.Any(container => container.TryAddAnimal(animal));
}
```

Hier voeg ik de animals aan de train toe.

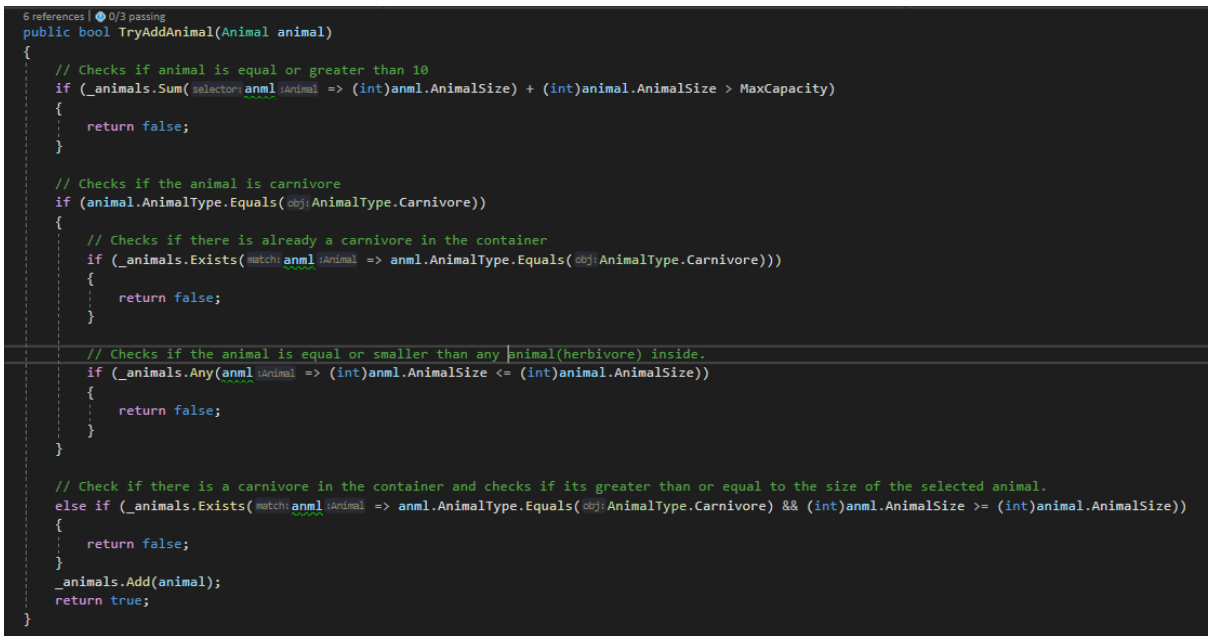
Hier controleer ik of de animal in een van de bestaande containers past en wanneer het niet in een bestaande container past, er een nieuwe container aangemaakt wordt.

```
[TestClass]
0 references
public class TrainTest
{
    [TestMethod]
    | 0 references
    public void AfterTheFirstContainerReachesMaximumCapacity_ANewContainerShouldBeCreated_Successfully()
    {
        //Arrange
        Train t = new Train();
        t.AddAnimalToTrain(new Animal(AnimalSize.Small, AnimalType.Herbivore));
        t.AddAnimalToTrain(new Animal(AnimalSize.Small, AnimalType.Herbivore));
        t.AddAnimalToTrain(new Animal(AnimalSize.Medium, AnimalType.Herbivore));
        t.AddAnimalToTrain(new Animal(AnimalSize.Large, AnimalType.Herbivore));


        //Act
        // First container is full, makes new container because the first container has reached maximum-capacity
        t.AddAnimalToTrain(new Animal(AnimalSize.Medium, AnimalType.Herbivore));

        //Assert
        Assert.AreEqual( expected: 2,  actual: t.Containers.Count);
    }
}
```

In deze Unittest check ik of de eerste container over de capaciteit gaat en er een nieuwe container succesvol wordt aangemaakt.

In de arrange Initialiseer ik de Train, en in de container met een gewicht van 10
In de act probeer ik een nieuwe animal (medium herbivore)toe te voegen met een gewicht van 3.
In de assert controleer ik of er een nieuwe container aangemaakt wordt door de expected value 2 te vergelijken met de container count.

Container

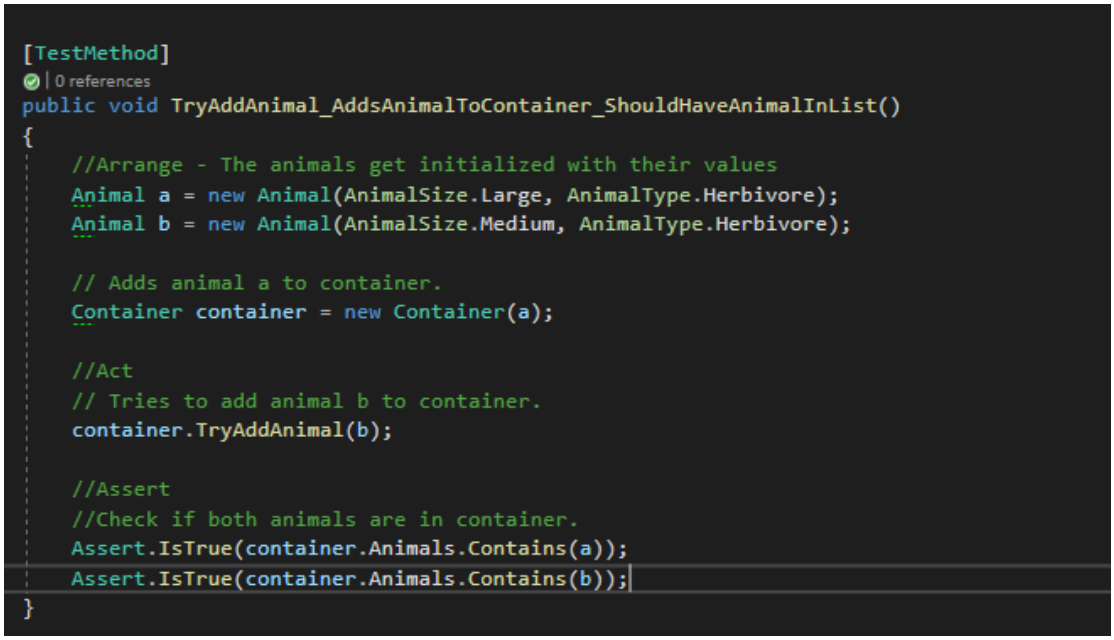


```
6 references | 0/3 passing
public bool TryAddAnimal(Animal animal)
{
    // Checks if animal is equal or greater than 10
    if (_animals.Sum(selector: anml :Animal => (int)anml.AnimalSize) + (int)animal.AnimalSize > MaxCapacity)
    {
        return false;
    }

    // Checks if the animal is carnivore
    if (animal.AnimalType.Equals(obj: AnimalType.Carnivore))
    {
        // Checks if there is already a carnivore in the container
        if (_animals.Exists(match: anml :Animal => anml.AnimalType.Equals(obj: AnimalType.Carnivore)))
        {
            return false;
        }

        // Checks if the animal is equal or smaller than any animal(herbivore) inside.
        if (_animals.Any(anml :Animal => (int)anml.AnimalSize <= (int)animal.AnimalSize))
        {
            return false;
        }
    }

    // Check if there is a carnivore in the container and checks if its greater than or equal to the size of the selected animal.
    else if (_animals.Exists(match: anml :Animal => anml.AnimalType.Equals(obj: AnimalType.Carnivore) && (int)anml.AnimalSize >= (int)animal.AnimalSize))
    {
        return false;
    }
    _animals.Add(animal);
    return true;
}
```

In de TryAddAnimal kijk ik of de animal toegevoegd kan worden aan de container.
In de functie controleer ik eerst of de animal gelijk of groter is dan 10 in de volgende if statement kijk ik of de animal van een animaltype, carnivore is.

Zoja dan kijk ik of er al een carnivore in de container zit. (zoja return false).

Hierna kijk ik of er een animal gelijk aan of kleiner is dan een animal binnen de container. (zoja return false). In de else if kijk ik of er een carnivore in de container zit en controleer ik of de carnivore in de container groter is of gelijk aan de geselecteerde animal.



```
[TestMethod]
0 references
public void TryAddAnimal_AddsAnimalToContainer_ShouldHaveAnimalInList()
{
    //Arrange - The animals get initialized with their values
    Animal a = new Animal(AnimalSize.Large, AnimalType.Herbivore);
    Animal b = new Animal(AnimalSize.Medium, AnimalType.Herbivore);

    // Adds animal a to container.
    Container container = new Container(a);

    //Act
    // Tries to add animal b to container.
    container.TryAddAnimal(b);

    //Assert
    //Check if both animals are in container.
    Assert.IsTrue(container.Animals.Contains(a));
    Assert.IsTrue(container.Animals.Contains(b));
}
```

In deze unittest kijk ik of er daadwerkelijk animals worden toegevoegd aan de lijst.
In animal a zit Large carnivore en bij animal b zit Medium Herbivore.

```
[TestMethod]
0 references
public void AddsAnimalToContainer_ChecksOverMaximumCapacity_ShouldResultFalse()
{
    //Arrange - The animals get initialized with their values
    Animal a = new Animal(AnimalSize.Large, AnimalType.Herbivore);
    Animal b = new Animal(AnimalSize.Medium, AnimalType.Carnivore);
    Animal c = new Animal(AnimalSize.Large, AnimalType.Herbivore);
    // Adds animal a to container.
    Container container = new Container(a);

    //Act
    // Adds animal b to container because there is a larger sized Animal in the same container so the carnivore doesn't eat it
    // and maximum-capacity has not been reached.
    container.TryAddAnimal(b);

    //Assert
    // Adds animal c to container.
    // Should result in False, because the maximum capacity has been reached within the same container.
    Assert.AreEqual( expected: container.TryAddAnimal(c),  actual: false);

}
```

In deze unittest test ik of de animals niet over de max capacity (10) kunnen gaan.

Ik add animal a in de arrange. Gewicht van de container: 5

In de act add ik animal b bij de container. Gewicht van de container: 8

In de assert vergelijk ik of ik animal c toe voegen kan voege met een waarde van 5. Aangezien de waarde van de container groter wordt dan 10, moet de functie false (aangezien de functie een bool is) teruggeven.

```
[TestMethod]
0 references
public void AddsAnimalToContainer_ChecksCarnivoreEatingBehaviour_ShouldResultFalse()
{
    //Arrange - The animals get initialized with their values
    Animal a = new Animal(AnimalSize.Large, AnimalType.Herbivore);
    Animal b = new Animal(AnimalSize.Medium, AnimalType.Carnivore);
    Animal c = new Animal(AnimalSize.Small, AnimalType.Herbivore);
    // Adds animal a to container.
    Container container = new Container(a);

    //Act
    // Adds animal b to container because there is a larger sized Animal in the same container so the carnivore doesn't eat it
    // and maximum-capacity has not been reached.
    container.TryAddAnimal(b);

    //Assert

    // Tries to add animal c to container
    // Should result in False, because there is a medium sized carnivore which will eat the smaller sized herbivore animal.
    Assert.AreEqual( expected: container.TryAddAnimal(c),  actual: false);
}
```

In deze unittest voeg ik een animal toe aan mijn container en controleer ik of het eetgedrag van de animal een false result in het toevoegen aan de container, wanneer van toepassing.

In de arrange add ik animal a (Large Herbivore), aan de container,

In de act, add ik animal b. Omdat animal a een Large Herbivore is, kan de Medium sized Carnivore erbij omdat een kleinere Carnivore geen grotere Herbivore kan eten. Het capacity van de container is nu 8.

Om te kijken of de container nou checkt op het eetgedrag controleer ik of ik nu nog een kleine carnivore kan toevoegen bij de medium sized carnivore(wat niet moet kunnen). Hierom kijk ik of de functie false returned wanneer de kleine carnivore wordt toegevoegd.