using System;

namespace CarManager

{

public class Car

{

private string make;

private string model;

private double fuelConsumption;

private double fuelAmount;

private double fuelCapacity;

private Car()

{

this.FuelAmount = 0;

}

public Car(string make, string model, double fuelConsumption, double fuelCapacity) : this()

{

this.Make = make;

this.Model = model;

this.FuelConsumption = fuelConsumption;

this.FuelCapacity = fuelCapacity;

}

public string Make

{

get

{

return this.make;

}

private set

{

if (String.IsNullOrEmpty(value))

{

throw new ArgumentException("Make cannot be null or empty!");

}

this.make = value;

}

}

public string Model

{

get

{

return this.model;

}

private set

{

if (String.IsNullOrEmpty(value))

{

throw new ArgumentException("Model cannot be null or empty!");

}

this.model = value;

}

}

public double FuelConsumption

{

get

{

return this.fuelConsumption;

}

private set

{

if (value <= 0)

{

throw new ArgumentException("Fuel consumption cannot be zero or negative!");

}

this.fuelConsumption = value;

}

}

public double FuelAmount

{

get

{

return this.fuelAmount;

}

private set

{

if (value < 0)

{

throw new ArgumentException("Fuel amount cannot be negative!");

}

this.fuelAmount = value;

}

}

public double FuelCapacity

{

get

{

return this.fuelCapacity;

}

private set

{

if (value <= 0)

{

throw new ArgumentException("Fuel capacity cannot be zero or negative!");

}

this.fuelCapacity = value;

}

}

public void Refuel(double fuelToRefuel)

{

if (fuelToRefuel <= 0)

{

throw new ArgumentException("Fuel amount cannot be zero or negative!");

}

this.FuelAmount += fuelToRefuel;

if (this.FuelAmount > this.FuelCapacity)

{

this.FuelAmount = this.FuelCapacity;

}

}

public void Drive(double distance)

{

double fuelNeeded = (distance / 100) \* this.FuelConsumption;

if (fuelNeeded > this.FuelAmount)

{

throw new InvalidOperationException("You don't have enough fuel to drive!");

}

this.FuelAmount -= fuelNeeded;

}

}

}

using System;

using CarManager;

using NUnit.Framework;

namespace Tests

{

public class CarTests

{

private Car car;

[SetUp]

public void Setup()

{

car = new Car("Toyota", "Verso", 10, 100);

}

[Test]

[TestCase("", "Verso", 5, 100)]

[TestCase(null, "Verso", 5, 100)]

[TestCase("Toyota", "", 5, 100)]

[TestCase("Toyota", null, 5, 100)]

[TestCase("Toyota", "Verso", 0, 100)]

[TestCase("Toyota", "Verso", -1, 100)]

[TestCase("Toyota", "Verso", 5, 0)]

[TestCase("Toyota", "Verso", 5, -10)]

public void Ctor\_ThrExInvalidData(string make, string model, double fuelConsumption, double fuelCapacity)

{

Assert.Throws<ArgumentException>((() => new Car(make, model, fuelConsumption, fuelCapacity)));

}

[Test]

public void Ctor\_SetValidData()

{

string make = "Toyota";

string model = "Verso";

double fuelConsumption = 5;

double fuelCapacity = 100;

car = new Car(make, model, fuelConsumption, fuelCapacity);

Assert.That(car.Make, Is.EqualTo(make));

Assert.That(car.Model, Is.EqualTo(model));

Assert.That(car.FuelConsumption, Is.EqualTo(fuelConsumption));

Assert.That(car.FuelCapacity, Is.EqualTo(fuelCapacity));

}

[Test]

[TestCase(0)]

[TestCase(-1)]

public void Refuel\_ThrExFuelIsZeroOrNegative(double fuelamount)

{

Assert.Throws<ArgumentException>(() => car.Refuel(fuelamount));

}

[Test]

public void Refuel\_IncreaseFuelamount()

{

double refuelAmount = car.FuelCapacity / 2;

car.Refuel(refuelAmount);

Assert.That(car.FuelAmount, Is.EqualTo(refuelAmount));

}

[Test]

public void Refuel\_SetFuelamountToCapacity()

{

car.Refuel(car.FuelCapacity \* 2);

Assert.That(car.FuelAmount, Is.EqualTo(car.FuelCapacity));

}

[Test]

public void Drive\_ThrExZeroFuel()

{

Assert.Throws<InvalidOperationException>(() => car.Drive(100));

}

[Test]

public void Drive\_EnoughFuel()

{

double fuel = car.FuelCapacity;

car.Refuel(fuel);

car.Drive(100);

Assert.That(car.FuelAmount, Is.EqualTo(fuel - car.FuelConsumption));

}

[Test]

public void Drive\_EnoughFuelForAllDistance()

{

car.Refuel(car.FuelCapacity);

double distance = car.FuelCapacity \* car.FuelConsumption;

car.Drive(distance);

Assert.That(car.FuelAmount, Is.EqualTo(0));

}

[Test]

public void FuelAmount\_ThrExNegativeValue()

{

car.Refuel(car.FuelCapacity);

double beforeDrive = car.FuelAmount;

car.Drive(100);

double afterDrive = car.FuelAmount;

Assert.That(afterDrive, Is.LessThan(beforeDrive));

}

}

}