Буниф Мажда

18-КБ-ПР2

**Отчет КПО Lab 9**

1) Задание выполняется совместно несколькими студентами (2-3 человека в группе).

2) Создать программу, реализующую функциональность, описанную в варианте задания из пункта 4. При разработке использовать паттерн Decorator. Результаты можно выводить в виде отформатированного текста с моноширинным шрифтом (консольное приложение).

3) Составить тесты к методам класса, проверяющие корректность их работы.

4) Запустить тестирование и проверить результаты работы.

5) Написать программу, демонстрирующую работу класса, осуществляющее визуальное представление результатов.

6) Составить отчет.

**4 Варианты заданий**

1) Печать чека с возможностью выборочного добавления/удаления его частей: название фирмы, адрес, рамка, данные чека (обязательная часть), рамка для обязательной части, сумма, рекламное объявление. Возможность выбора реализовать через CheckBox’ы.

**Листинг :**

**Component.cs**

public interface ReceiptCarComponent

{

string Compagny\_Name();

string Address();

string GetDescription();

double GetCost();

}

**ConcreteComponent.cs**

public class EconomyCar : ReceiptCarComponent

{

public string Compagny\_Name()

{

return " BMW";

}

public string Address()

{

return "Allemand 50 zurich";

}

public string GetDescription()

{

return " Economy Car";

}

public double GetCost()

{

return 450000;

}

}

public class DeluxCar : ReceiptCarComponent

{

public string Compagny\_Name()

{

return " Mercedes";

}

public string Address()

{

return "USA 50 zurich";

}

public string GetDescription()

{

return " Delux Car";

}

public double GetCost()

{

return 750000.0;

}

}

public class LuxuryCar : ReceiptCarComponent

{

public string Compagny\_Name()

{

return " Lamborghini";

}

public string Address()

{

return "Japan xx5js";

}

public string GetDescription()

{

return " Luxury Car";

}

public double GetCost()

{

return 1000000.0;

}

}

}

**Decorator.cs**

public abstract class CarAccessoriesDecorator : ReceiptCarComponent

{

/// <summary>

/// here is the abstract decorator

/// </summary>

private ReceiptCarComponent car;

public CarAccessoriesDecorator(ReceiptCarComponent aCar)

{

this.car = aCar;

}

public virtual string Compagny\_Name()

{

return this.car.Compagny\_Name();

}

public virtual string Address()

{

return this.car.Address();

}

public virtual string GetDescription()

{

return this.car.GetDescription();

}

public virtual double GetCost()

{

return this.car.GetCost();

}

}

/// <summary>

/// Concrete Decorator A

/// </summary>

public class BasicAccessories : CarAccessoriesDecorator

{

public BasicAccessories(ReceiptCarComponent aCar) : base(aCar)

{

}

public override string Compagny\_Name()

{

return base.Compagny\_Name();

}

public override string Address()

{

return base.Address();

}

public override string GetDescription()

{

return base.GetDescription() + " Basic accessories package";

}

public override double GetCost()

{

return base.GetCost() + 6000.0;

}

}

/// Concrete Decorator B

public class AdvancedAccessories : CarAccessoriesDecorator

{

public AdvancedAccessories(ReceiptCarComponent aCar) : base(aCar)

{

}

public override string Compagny\_Name()

{

return base.Compagny\_Name();

}

public override string Address()

{

return base.Address();

}

public override string GetDescription()

{

return base.GetDescription() + " Advanced accessories package";

}

public override double GetCost()

{

return base.GetCost() + 10000.0;

}

}

/// Concrete Decorator C

public class SportsAccessories : CarAccessoriesDecorator

{

public SportsAccessories(ReceiptCarComponent aCar) : base(aCar)

{

}

public override string Compagny\_Name()

{

return base.Compagny\_Name();

}

public override string Address()

{

return base.Address();

}

public override string GetDescription()

{

return base.GetDescription() + " Sports accessories package";

}

public override double GetCost()

{

return base.GetCost() + 15000.0;

}

}

}

**Tests.cs**

namespace ClassLibrary1

{

[TestFixture]

public class Component

{

[Test]

public void TestEconomyCar()

{

ReceiptCarComponent car = new EconomyCar();

CarAccessoriesDecorator acc = new BasicAccessories(car);

string actual1 = acc.GetDescription();

string expected1 = " Economy Car Basic accessories package";

Assert.AreEqual(actual1, expected1);

}

[Test]

public void TestDeluxCar()

{

ReceiptCarComponent car = new DeluxCar();

CarAccessoriesDecorator acc = new BasicAccessories(car);

string actual1 = acc.GetDescription();

string expected1 = " Delux Car Basic accessories package";

Assert.AreEqual(actual1, expected1);

}

[Test]

public void TestluxCar()

{

ReceiptCarComponent car = new LuxuryCar();

CarAccessoriesDecorator acc = new BasicAccessories(car);

string actual1 = acc.GetDescription();

string expected1 = " Luxury Car Basic accessories package";

Assert.AreEqual(actual1, expected1);

}

[Test]

public void TestCostCars()

{

ReceiptCarComponent car1 = new DeluxCar();

CarAccessoriesDecorator acc1 = new AdvancedAccessories(car1);

ReceiptCarComponent car2= new LuxuryCar();

CarAccessoriesDecorator acc2 = new SportsAccessories(car2);

ReceiptCarComponent car3 = new EconomyCar();

CarAccessoriesDecorator acc3 = new BasicAccessories(car3);

var actual1 = acc1.GetCost();

double expected1 = 760000.0;

var actual2 = acc2.GetCost();

double expected2 = 1015000.0;

var actual3 = acc3.GetCost();

double expected3 = 456000.0;

Assert.AreEqual(actual1, expected1);

Assert.AreEqual(actual2, expected2);

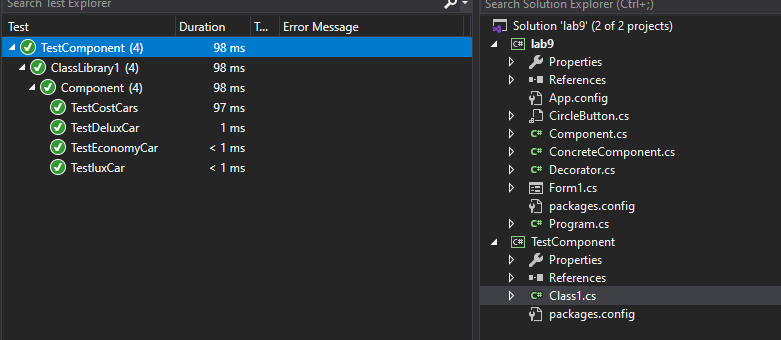
Assert.AreEqual(actual3, expected3);

}

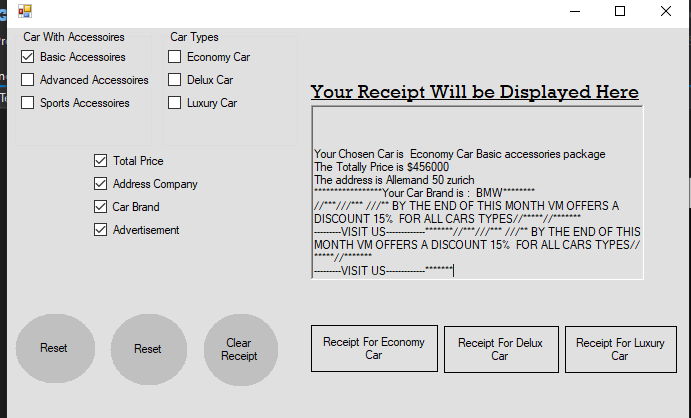
}

}

**Результат Работы**



***Add element to The Receipt***



***Delete element From the receipt***

