МИНОБРНАУКИ РФ

ФГБОУ ВО Тверской государственный технический университет

Кафедра “Программное обеспечение”

**Проект по теме: «Классы. Ссылочный тип данных.»**

**Класс Tile.**

Выполнила:

Команда №11, 2 курс, факультет

информационных технологий, группа: Б.ПИН.РИС.20.05

Проверил: Биллиг Владимир Арнольдович

Тверь, 2022

Оглавление

[1. Постановка задачи 3](#_Toc99177984)

[2. Назначение проекта Tile 3](#_Toc99177985)

[3. Класс Tile 3](#_Toc99177986)

[4. Тестирование класса Tile 7](#_Toc99177987)

[5. Интерфейс Tile Shop 11](#_Toc99177988)

[6. Тестирование интерфейса Tile Shop 16](#_Toc99177989)

# ****Постановка задачи****

**Разработать класс Tiles. Помимо этого, нужно:**

* **создать консольный проект, где будет проводиться тестирование класса;**
* **построить интерфейсный проект (форму) для пользователя, чтобы он смог рассмотреть работу класса. Построим форму «Магазин керамических плиток», который будет имитировать работу реального магазина керамических плиток.**

# ****Назначение проекта Tile****

Классы нужны для создания своих структур данных, которые будут содержать какую-либо логику обработки. Вся логика хранится в описании класса, при этом оставляя в вызывающей программе лаконичные вызовы, без лишнего кода.

Главной задачей нашего проекта является рассмотрение ссылочного типа данных (класс). **Так,** благодаря нашему проекту любой начинающий программист сможет рассмотреть работу класса (ссылочного типа), и также рассмотреть реализацию магазина плиток. Для реализации этого магазина плиток будет использоваться один класс, задающий объект «Плитка».

Дальнейшую реализацию «Магазина керамических плиток» может использовать, например, какая-нибудь небольшая компания по продаже керамических плиток, но необходимо будет брать всю информацию о плитках из базы данных, а не генерировать её как в примере.

Помимо, этого **можно будет рассмотреть использование класса как типа данных.**

# Класс Tile

1. Теория по классам

Для начала разберём определение класса и его назначение.

**Класс** — это описание того, какими свойствами и поведением будет обладать объект. А сам **объект** — это экземпляр с собственным состоянием этих свойств. В свою очередь класс имеет 2 роли:

1. **Класс — это тип данных**, задающий реализацию некоторой абстракции данных, характерной для проблемной области, в интересах которой создается программная система.
2. **Класс — это модуль**, архитектурная единица построения программной системы. Модульность построения - основное свойство программных систем. Имеет модификатор static.

**Синтаксис класса:**

[атрибуты] [модификаторы] class имя\_класса :родители\_класса

{тело класса}

Наш класс будет в роли типа данных.

**Поля и методы**

При определении полей мы можем присвоить им некоторые значения. Если поля класса не инициализированы, то они получают значения по умолчанию.

Методы класса имеют доступ к его полям. И чтобы эти методы были доступны вне класса, они должны быть с модификатором public.

**Создание объекта класса:**

*Имя\_Класса название\_переменой = new конструктор\_класса(параметры\_конструктора);*

**Обращение к функциональности класса**

Для обращения к функциональности класса - полям, методам (а также другим элементам класса) применяется точечная нотация точки - после объекта класса ставится точка, а затем элемент класса:

*объект.**поле\_класса*

*объект.метод\_класса(параметры\_метода)*

*имя\_класса.метод\_класса(параметры\_метода)*

*имя\_класса.поле\_класса*

Теперь нам осталось разобраться с ссылочным типом данных.

1. Теория по ссылочному типу данных

**!** *Для создания класса развёрнутого типа создаётся struct(структура), а для создания класса ссылочного типа создаётся class(класс).*

Если *класс* T относится к **развернутому типу**, то в стеке сущности x отводится *память*, необходимая объекту класса T. Говорят, что *объект* разворачивается на памяти, жестко связанной с сущностью x. Эту *память* сущность x ни с кем не разделяет.

**! *Стек*** *— это область оперативной памяти, которая создаётся для каждого потока.*

Если *класс* T относится к **ссылочному типу**, то *память* объекту отводится в *куче*, а в стеке сущности x отводится дополнительная *память*, содержащая ссылку на *объект*.

**!Куча** — это хранилище памяти, также расположенное в ОЗУ, которое допускает динамическое выделение памяти и не работает по принципу стека: это просто склад для ваших переменных. Когда вы выделяете в куче участок памяти для хранения переменной, к ней можно обратиться не только в потоке, но и во всем приложении.

**!** *Сущности x в стеке всегда отводится память, но какая память - зависит от того, к развернутому или ссылочному типу относится класс Т.*

1. Реализация класса Tile

Создадим класс Tile:

/// <summary>

/// Класс "Плитка"

/// Помогает рассчитать

/// количество упаковок плиток на пространство

/// </summary>

public class Tile

{}

Класс будет иметь модификатор доступа public, так как будет использоваться в форме и в консоли.

Добавим следующие поля класса:

Brand brand;//бренд плитки

public double width, length, price\_packaging;//ширина,длина,цена упаковки

int quantity\_packaging;//кол-во плиток в упаковке

Как мы видим в классе будет использовано перечисление Brand. Это сделано для того, чтобы упростить задачу по инициализации бренда для плитки, так как, например, если бренд будет реализован строкой, то тяжелее будет генерировать случайную плитку для магазина. А так мы будет использовать биты, что упростит нам задачу по генерации плиток. Перечисление Brand:

/// <summary>

/// Перечисление брендов

/// </summary>

public enum Brand { ARTENS =0,AZORI =1,BELMAR = 2,CULTO = 3,

ESTIMA=4,MAINZU=5 };

Создание конструктора класса

У нас будет один конструктор с параметрами, который будет инициализировать все поля, указанные выше.

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="brand">бренд</param>

/// <param name="width">ширина</param>

/// <param name="length">длина</param>

/// <param name="price\_packaging">цена упаковки</param>

/// <param name="quantity\_packaging">количество плиток в упаковке</param>

public Tile(Brand brand, double width, double length, double price\_packaging, int quantity\_packaging)

{

this.brand = brand;

this.width = width;

this.length = length;

this.price\_packaging = price\_packaging;

this.quantity\_packaging = quantity\_packaging;

}

Нет необходимости прописывать конструктор по умолчанию.

Переопределение метода ToString

Этот метод будет необходим для вывода полной информации о плитке.

/// <summary>

/// Вывод информации о плитке

/// </summary>

/// <returns>Строка со всей информацией о плитке</returns>

public override string ToString()

{

string res = "Кафельная плитка " + brand.ToString() +

"\nРазмер: " + length + " x " + width +

" кв.см.\nВ 1 упаковке "+ Math.Round(HowMetersInPackage(),3) + " кв.м.\nЦена за упаковку (" + quantity\_packaging + " шт) : " + price\_packaging + " р.";

return res ;

}

Тут также выводиться информация об количестве квадратных метров в одной упаковке.

Метод HowMetersInPackage

Этот метод рассчитывает количество квадратных метров в одной упаковке плитки, используя следующую формулу:

.

/// <summary>

/// Рассчитывает количество кв.м. в 1 упаковке

/// </summary>

/// <returns>количество кв.м. в 1 упаковке</returns>

private double HowMetersInPackage()

{

return (width / 100 \* length / 100) \* quantity\_packaging;

}

Далее этот метод будем использовать для расчёта упаковок на пространство.

Стоит отметить, что метод HowMetersInPackage имеет модификатор доступа private, так как является вспомогательным для класса.

Метод NumberOfPackage

Он рассчитывает количество упаковок на пространство. У него будет один параметр (площадь пространства).

/// <summary>

/// Рассчитывает количество упаковок на пространство

/// </summary>

/// <param name="square">площадь пространства в кв.м.</param>

/// <returns>количество упаковок на пространство</returns>

public int NumberOfPackages(double square)

{

double res = square / HowMetersInPackage();

if (square % HowMetersInPackage() == 0)

return int.Parse(res.ToString());

else

return int.Parse(Math.Truncate(res).ToString()) + 1;

}

Если площадь пространства делиться на кол-во кв.м. в упаковке на цело,

то выводиться результат этого деления,

иначе выводиться целая часть результата этого деления, увеличенная на 1.

Метод Price

Он рассчитывает цену упаковок, используя формулу:

/// <summary>

/// Рассчитывает цену за все упаковки

/// </summary>

/// <param name="packages">количество упаковок</param>

/// <returns>Цена за все упаковки</returns>

public double Price(int packages)

{

return Math.Round(packages \* price\_packaging,2);

}

# Тестирование класса Tile

Создадим консольный проект для тестирования класса и напишем пару примеров создания плитки (применяем обращение к функциональности класса):

Тест №1:

/// <summary>

/// Тестирование класса Tiles

/// плитку вводим самостоятельно в коде

/// </summary>

static void Test1()

{

Tile t = new Tile((Brand)1, 40, 40, 234, 23);

Console.WriteLine(t.ToString());

Console.WriteLine("\n-ОЖИДАЕМЫЕ ДАННЫЕ:");

double m = 100, m1 = 16;

string res = "На пространство в " + m + " м^2 потребуется: 625 плиток.\n"+

"На пространство в " + m + " м^2 потребуется: 28 упаковок.\n"+

"Цена всех упаковок: 6 552 р.\n"+

"На пространство в " + m1 + " м^2 потребуется: 100 плиток.\n" +

"На пространство в " +m1 + " м^2 потребуется: 5 упаковок.\n" +

"Цена всех упаковок: 1 170 р.\n";

Console.WriteLine(res);

Console.WriteLine("-РАБОТА КЛАССА: ");

Console.WriteLine("Кол-во упаковок этой плитки на пространство в " + m + " м^2: " + t.NumberOfPackages(m));

Console.WriteLine("Цена всех упаковок: " + t.Price(t.NumberOfPackages(m)));

Console.WriteLine("Кол-во упаковок этой плитки на пространство в 4 x 4 м^2: " + t.NumberOfPackages(m1));

Console.WriteLine("Цена всех упаковок: " + t.Price(t.NumberOfPackages(m1)));

}

Результат консоли:

|  |  |
| --- | --- |
|  | Кафельная плитка AZORI  Размер: 40 x 40 кв.см.  В 1 упаковке 3,68 кв.м.  Цена за упаковку (23 шт) : 234 р.  -ОЖИДАЕМЫЕ ДАННЫЕ:  На пространство в 100 м^2 потребуется: 625 плиток.  На пространство в 100 м^2 потребуется: 28 упаковок.  Цена всех упаковок: 6 552 р.  На пространство в 16 м^2 потребуется: 100 плиток.  На пространство в 16 м^2 потребуется: 5 упаковок.  Цена всех упаковок: 1 170 р.  -РАБОТА КЛАССА:  Кол-во упаковок этой плитки на пространство в 100 м^2: 28  Цена всех упаковок: 6552  Кол-во упаковок этой плитки на пространство в 4 x 4 м^2: 5  Цена всех упаковок: 1170 |

Тест №2:

/// <summary>

///Тестирование класса Tiles

/// плитку вводит клиент самостоятельно в консоле

/// </summary>

static void Test2()

{

Console.Write("Введите ширину плитки: ");

double width = double.Parse(Console.ReadLine());

Console.Write("Введите длину плитки: ");

double length = double.Parse(Console.ReadLine());

Console.Write("Введите цену за упаковку плитки: ");

double price\_packaging = double.Parse(Console.ReadLine());//ширина,длина,цена упаковки

Console.Write("Введите кол-во штук в упаковке: ");

int quantity\_packaging = int.Parse(Console.ReadLine());

Random r = new Random();

Tile t = new Tile((Brand)r.Next(0, 5), width, length, price\_packaging, quantity\_packaging);

Console.WriteLine(t.ToString());

Console.Write("Введите ширину пространства: ");

double l1 = double.Parse(Console.ReadLine());

Console.Write("Введите длину пространства: ");

double l2 = double.Parse(Console.ReadLine());

double M1 = l1 \* 10, M2 = l1 \*l2 ;

double s\_tile = width \* length / 10000;

double res1\_tile = M1 / s\_tile;

if (res1\_tile % quantity\_packaging == 0)

res1\_tile /= quantity\_packaging;

else

res1\_tile= Math.Truncate(res1\_tile/ quantity\_packaging)+1;

string res = "На пространство в " + M1 + " м^2 потребуется: "+ res1\_tile+" упаковок.\n" +

"Цена всех упаковок: "+ res1\_tile\*price\_packaging+" р.\n";

double res2\_tile=M2 / s\_tile;

if (res2\_tile % quantity\_packaging == 0)

res2\_tile /= quantity\_packaging;

else

res2\_tile = Math.Truncate(res2\_tile / quantity\_packaging) + 1;

res += "На пространство в " + M2 + " м^2 потребуется: " + res2\_tile + " упаковок.\n" +

"Цена всех упаковок: " + res2\_tile \* price\_packaging + " р.\n";

Console.WriteLine("\n-ОЖИДАЕМЫЕ ДАННЫЕ:");

Console.WriteLine(res);

Console.WriteLine("-РАБОТА КЛАССА: ");

Console.WriteLine("Кол-во упаковок этой плитки на пространство в " + M1 + " м^2: " + t.NumberOfPackages(M1));

Console.WriteLine("Цена всех упаковок: " + t.Price(t.NumberOfPackages(M1)));

Console.WriteLine("Кол-во упаковок этой плитки на пространство в " + l1 + " x " + l2 + " м^2^: " + t.NumberOfPackages(M2));

Console.WriteLine("Цена всех упаковок: " + t.Price(t.NumberOfPackages(M2)));

}

Результат консоли:

|  |  |
| --- | --- |
|  | Введите ширину плитки: 34  Введите длину плитки: 45  Введите цену за упаковку плитки: 1234  Введите кол-во штук в упаковке: 33  Кафельная плитка ESTIMA  Размер: 45 x 34 кв.см.  В 1 упаковке 5,049 кв.м.  Цена за упаковку (33 шт) : 1234 р.  Введите ширину пространства: 3  Введите длину пространства: 6  -ОЖИДАЕМЫЕ ДАННЫЕ:  На пространство в 30 м^2 потребуется: 6 упаковок.  Цена всех упаковок: 7404 р.  На пространство в 18 м^2 потребуется: 4 упаковок.  Цена всех упаковок: 4936 р.  -РАБОТА КЛАССА:  Кол-во упаковок этой плитки на пространство в 30 м^2: 6  Цена всех упаковок: 7404  Кол-во упаковок этой плитки на пространство в 3 x 6 м^2^: 4  Цена всех упаковок: 4936 |

Тест №3:

/// <summary>

/// Тестирование класса Tiles

/// плитку вводим самостоятельно в коде

/// </summary>

static void Test3()

{

Tile t = new Tile((Brand)1, 50, 43, 234, 23);

Console.WriteLine(t.ToString());

Console.WriteLine("\n-ОЖИДАЕМЫЕ ДАННЫЕ:");

double m = 0.5, m1 = 0.5\*3;

string res =

"На пространство в " + m + " м^2 потребуется: 1 упаковка.\n" +

"Цена всех упаковок: 234 р.\n" +

"На пространство в " + m1 + " м^2 потребуется: 1 упаковка.\n" +

"Цена всех упаковок: 234 р.\n";

Console.WriteLine(res);

Console.WriteLine("-РАБОТА КЛАССА: ");

Console.WriteLine("Кол-во упаковок этой плитки на пространство в "+m+" м^2: " + t.NumberOfPackages(m));

Console.WriteLine("Цена всех упаковок: " + t.Price(t.NumberOfPackages(m)));

Console.WriteLine("Кол-во упаковок этой плитки на пространство в 0.5 x 3 м^2: " + t.NumberOfPackages(m1));

Console.WriteLine("Цена всех упаковок: " + t.Price(t.NumberOfPackages(m1)));

}

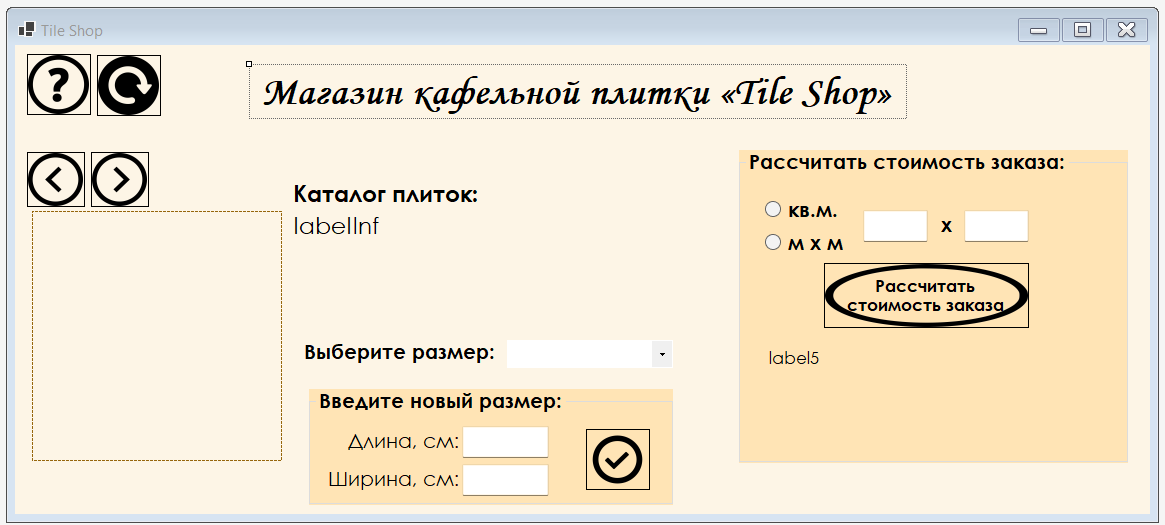
Результат консоли:

|  |  |
| --- | --- |
|  | Кафельная плитка AZORI  Размер: 43 x 50 кв.см.  В 1 упаковке 4,945 кв.м.  Цена за упаковку (23 шт) : 234 р.  -ОЖИДАЕМЫЕ ДАННЫЕ:  На пространство в 0,5 м^2 потребуется: 1 упаковка.  Цена всех упаковок: 234 р.  На пространство в 1,5 м^2 потребуется: 1 упаковка.  Цена всех упаковок: 234 р.  -РАБОТА КЛАССА:  Кол-во упаковок этой плитки на пространство в 0,5 м^2: 1  Цена всех упаковок: 234  Кол-во упаковок этой плитки на пространство в 0.5 x 3 м^2: 1  Цена всех упаковок: 234 |

# Интерфейс Tile Shop

1. Конструктор формы

Создадим форму. В конструкторе она выглядит так:



1. Расшифровка кнопок и элементов формы

|  |  |
| --- | --- |
|  | Кнопка «Справка»  Выводит информацию о магазине и о форме. |
|  | Кнопка «Обновления»  Обновляет интерфейс и создаёт новый массив с плитками. |
|  | Кнопка «Назад»  Возращение к предыдущей плитки. |
|  | Кнопка «Далее»  Переход к следующей плитки. |
|  | pictureBox  Вывод изображения плитки. |
|  | На label выводиться вся информация по плитке. |
|  | Позволяет выбрать размеры по умолчанию или ввести свой. |
|  | Ввод нового размера. |
|  | Расчёт плитки на пространство. |

1. Особенности интерфейса

* **Работа магазина:**
* Не осуществляется продажа плиток нецелых размеров;
* Размер вводимой плитки должен соответствовать диапазону:
* **Работа формы:**
* Ввод нового размера:
* При вводе размера, который не входит в вышеуказанный диапазон, текстовое поле будет окрашиваться красным цветом;
* Нельзя вводить буквы, пробел, точку, запятую и другие знаки;
* Если пользователь нажмёт кнопку OK с неверно введённым размером, то появиться окно с ошибкой.
* Ввод размера пространства:
* Можно выбрать вид ввода (произведение или готовый метраж);
* В текстовые поля ввода нельзя вводить буквы, пробел и все другие знаки, кроме точки и запятой.
* По умолчанию размер плитки 50 х 50 кв.см;
* При выборе размера плитки pictureBox будет менять свой размер, и при этом прочие объекты будут равномерно распределяться по форме;
* По умолчанию генерируется массив из 10 плиток;
* На 1 плитке кнопка «Назад» блокируется, на последней плитке кнопка «Далее» блокируется;
* Фотографии для плиток находятся в каталоге формы;
* Созданы два дополнительных класса для работы магазина;
* В форме применяется округление кнопок.

1. Дополнительный класс генерации плиток

Является статическим, так как нужен для «хранения участка кода» и декомпозиции. Рандомно генерирует плитки с размером по умолчанию 50 х 50 кв.см. Все, кроме одного метода (возвращает массив случайно сгенерированных плиток), закрыты от пользователя.

Класс является закрытым, так как нужен только для использования в форме.

/// <summary>

/// Класс генерирования массива плиток

/// </summary>

static class GenerateTiles

{

static Random r = new Random();

/// <summary>

/// Генерирование бренда

/// </summary>

/// <returns>бренд</returns>

static Brand GenerateBrand()

{

return (Brand)r.Next(0, 6);

}

/// <summary>

/// Генерирование цены

/// </summary>

/// <param name="k">кол-во плиток в упаковке</param>

/// <returns>цена</returns>

static double GeneratePrice(int k)

{

return k \*0.5\* r.Next(150, 500);

}

/// <summary>

/// Генерирование кол-ва штук в упаковке

/// </summary>

/// <returns>кол-во плиток</returns>

static int GenerateQuantity()

{

return r.Next(10, 25);

}

/// <summary>

/// Генерирование массива плиток

/// </summary>

/// <param name="len">длина массива</param>

/// <returns>массив плиток длиной len</returns>

public static Tiles[] GenerateArrTiles(int len)

{

Tiles[] arr = new Tiles[len];

for(int i = 0;i<len;i++)

{

int k = GenerateQuantity();

double price = GeneratePrice(k);

arr[i] = new Tiles(GenerateBrand(), 50, 50, price, k);

}

return arr;

}

}

1. Дополнительный класс для работы формы

Является статическим, так как нужен для декомпозиции работы формы. Класс является закрытым, так как нужен только для использования в форме.

/// <summary>

/// Класс работы формы

/// </summary>

static class ClassWorkForm1

{

/// <summary>

/// Генерирование стоимости плиток

/// в соответствии с размерами

/// </summary>

/// <param name="s">площадь плитки</param>

/// <param name="index">индекс плитки</param>

/// <param name="arrPrice">массив первоначальных цен</param>

/// <returns>сгенерированная стоимость</returns>

public static double GeneratedPriceTile(double s, int index, double[] arrPrice)

{

double price = arrPrice[index];

if (s == 2500)

return price;

return Math.Round(price \* (s / 2500), 2);

}

/// <summary>

/// Массив первоначальных цен

/// </summary>

/// <param name="ArrTiles">массив плиток</param>

/// <returns>массив цен</returns>

public static double[] arrPriceTile(Tiles [] ArrTiles)

{

double[] price = new double[ArrTiles.Length];

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

price[i] = ArrTiles[i].price\_packaging;

return price;

}

/// <summary>

/// Чтение картинок из каталога

/// </summary>

/// <returns>массив картинок</returns>

public static FileInfo[] arrJPG()

{

DirectoryInfo Dir = new DirectoryInfo(Directory.GetCurrentDirectory());

FileInfo[] Files = Dir.GetFiles("\*.jpg");

return Files;

}

/// <summary>

/// Массив картинок для плиток

/// </summary>

/// <param name="ArrTiles">массив плиток</param>

/// <returns>массив номеров картинок</returns>

public static int[] arrNumTile(Tiles[] ArrTiles)

{

int[] arr = new int[ArrTiles.Length];

Random r = new Random();

int i = 0;

while (i < arr.Length)

{

int u = r.Next(0, 13);

if (!Find(arr, u) || i > 13)

{

arr[i] = u;

i++;

}

}

return arr;

}

/// <summary>

/// Проверка на наличие в массиве элемента

/// </summary>

/// <param name="arr">массив значение</param>

/// <param name="n">значение, которое нужно проверить</param>

/// <returns>true - элемент есть в массиве,false-элемента нет в массиве</returns>

static bool Find(int[] arr, int n)

{

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

if (arr[i] == n)

return true;

return false;

}

/// <summary>

/// Множитель для размера картинки

/// </summary>

/// <param name="l">длина плитки</param>

/// <param name="w">ширина плитки</param>

/// <returns>множитель</returns>

public static int MultiplierSizePictureBox(double l, double w)

{

if (l > 50 || w > 50)

return 3;

else

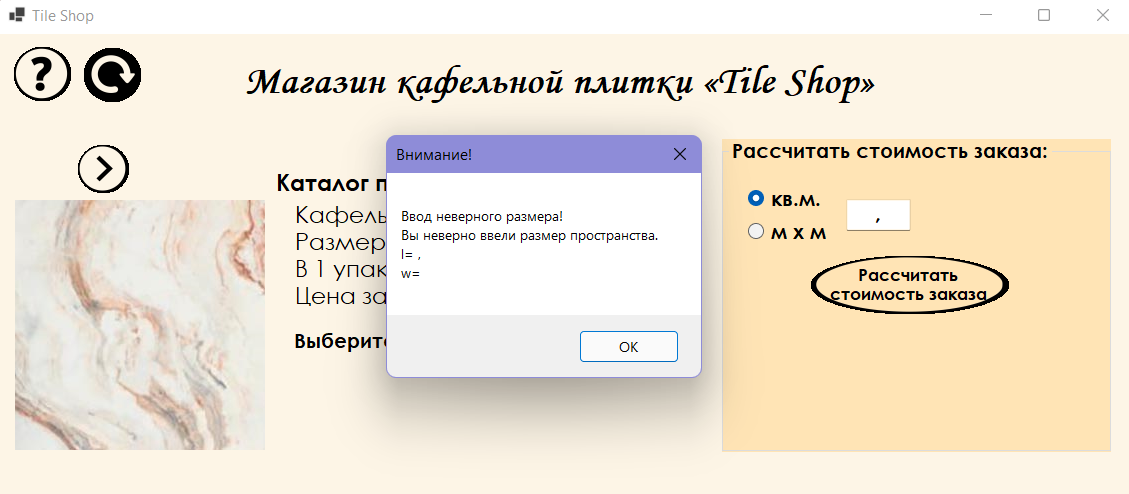
return 5;

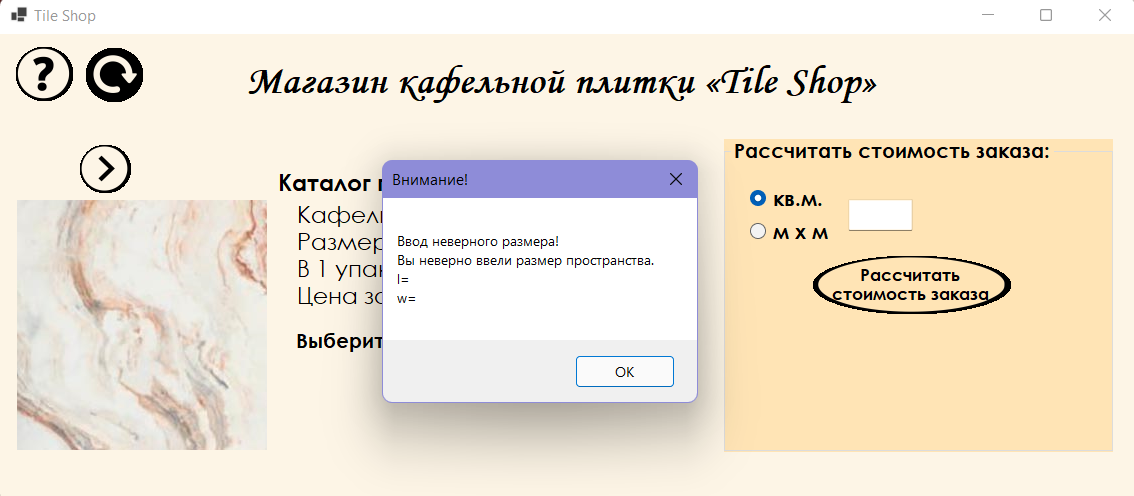
}

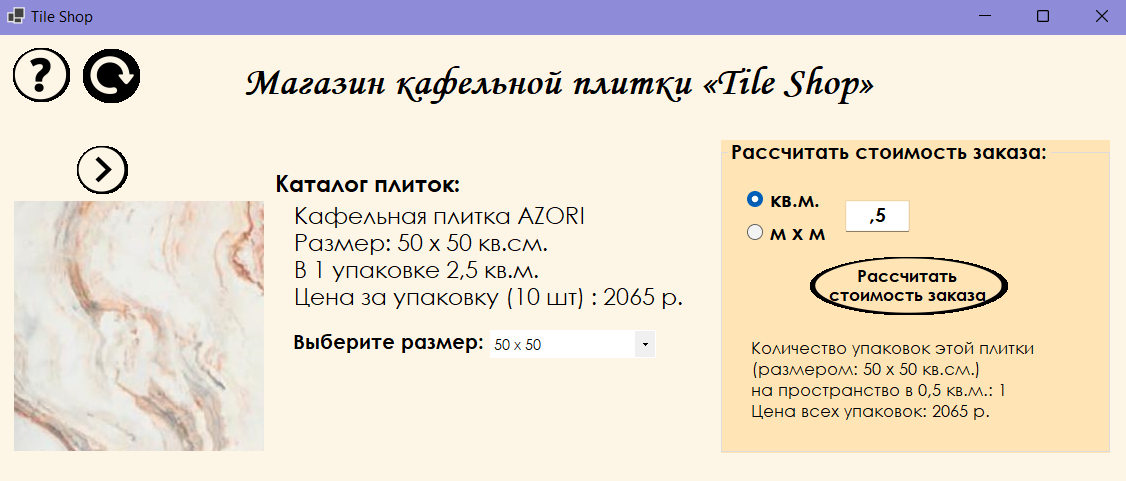
}

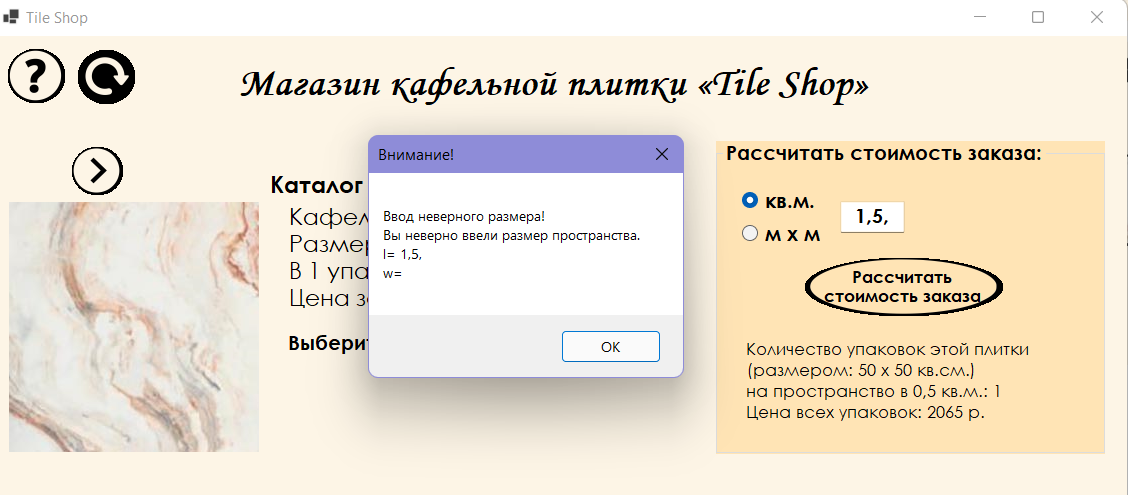
# Тестирование интерфейса Tile Shop

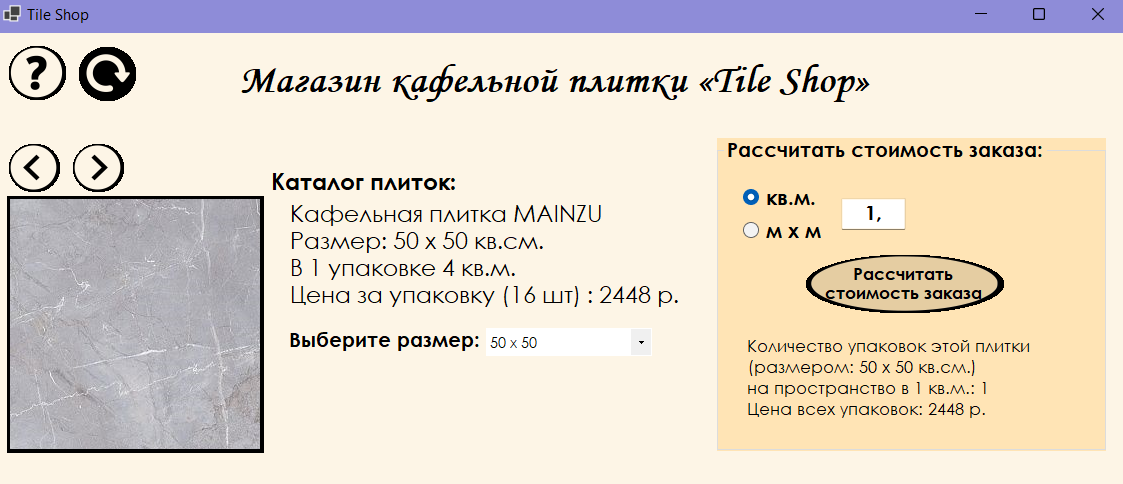
**Ситуации: ошибок:**

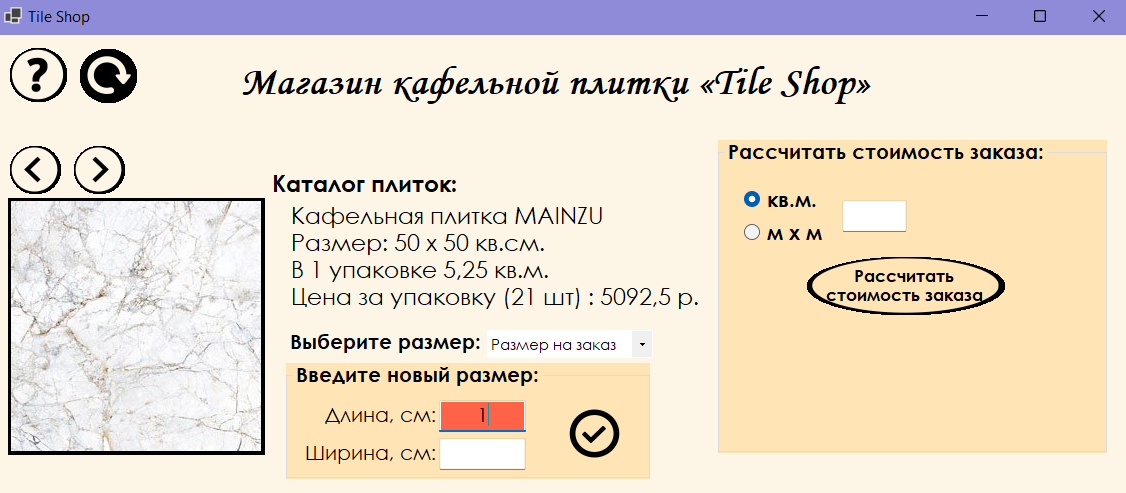




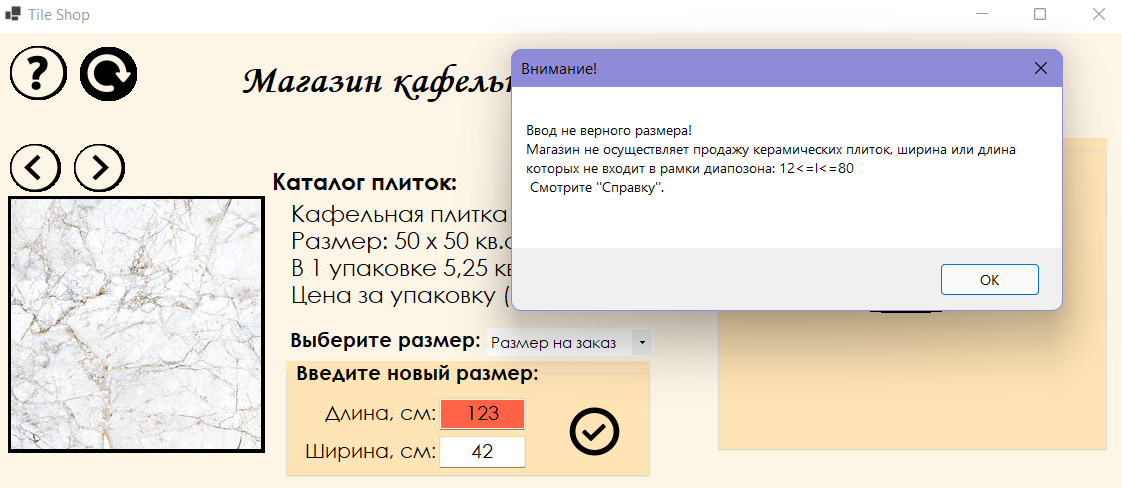




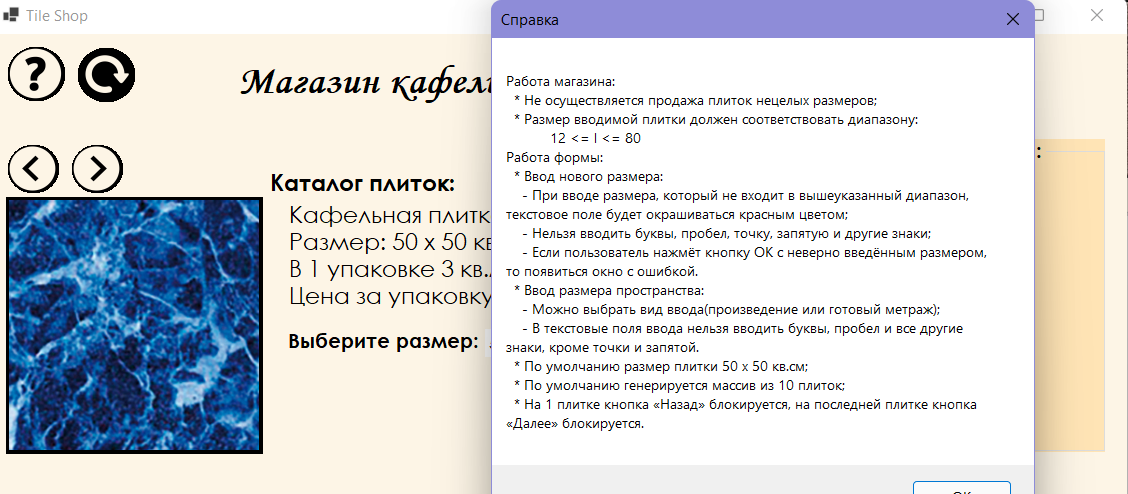




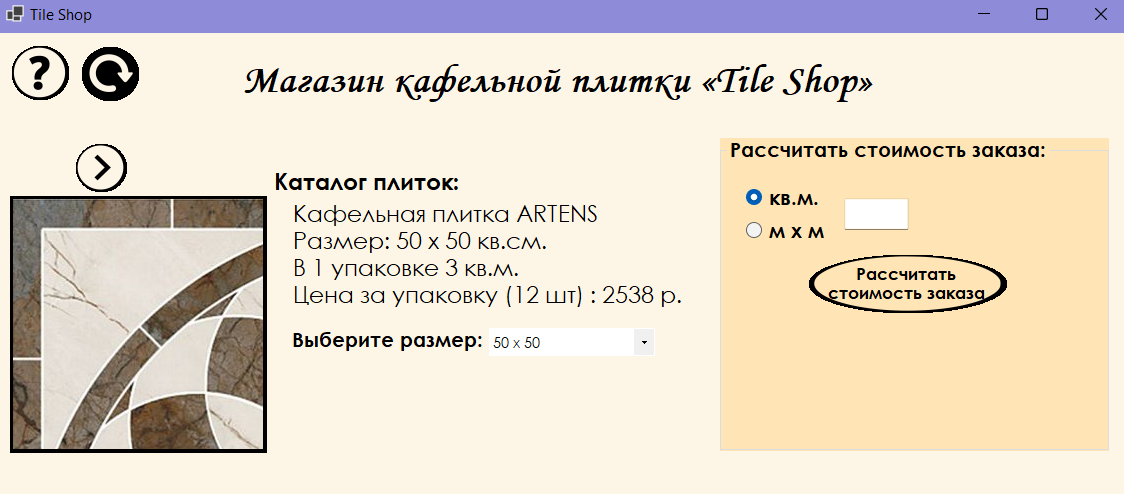


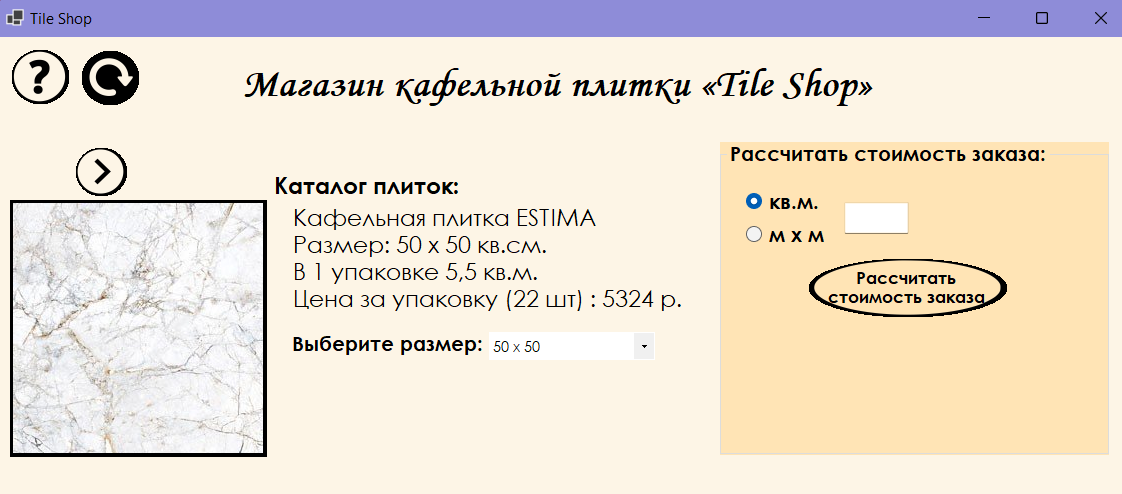


**Справка:**

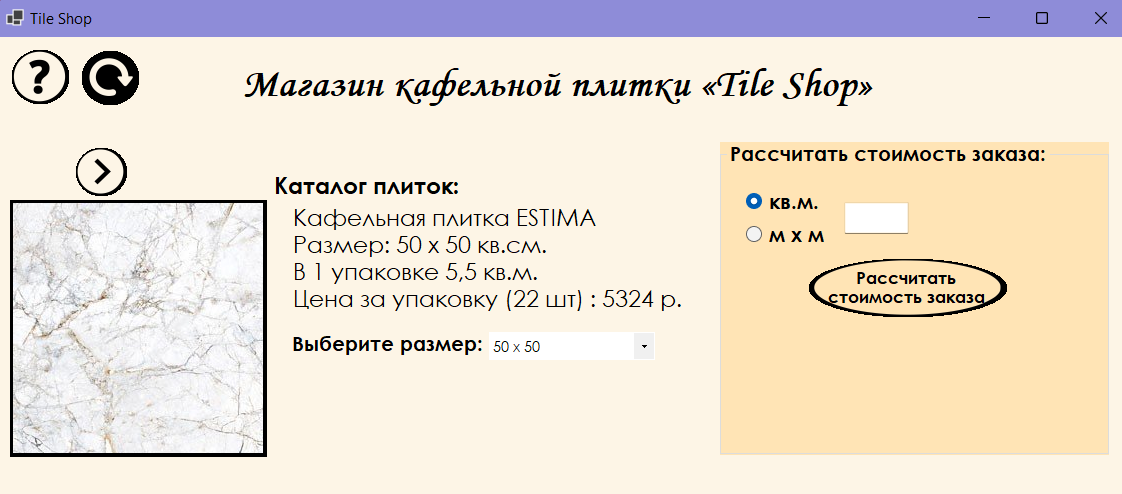


**Обновление:**

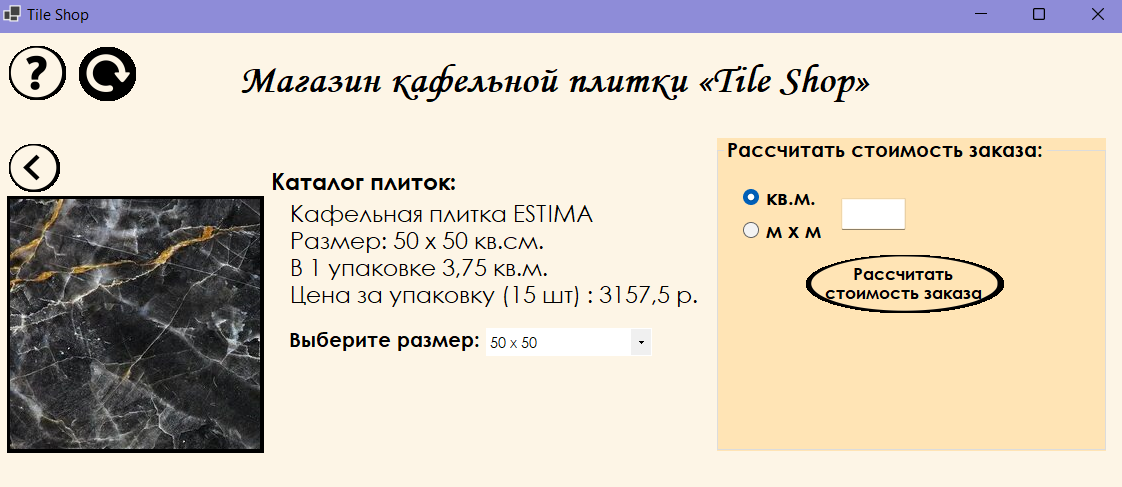




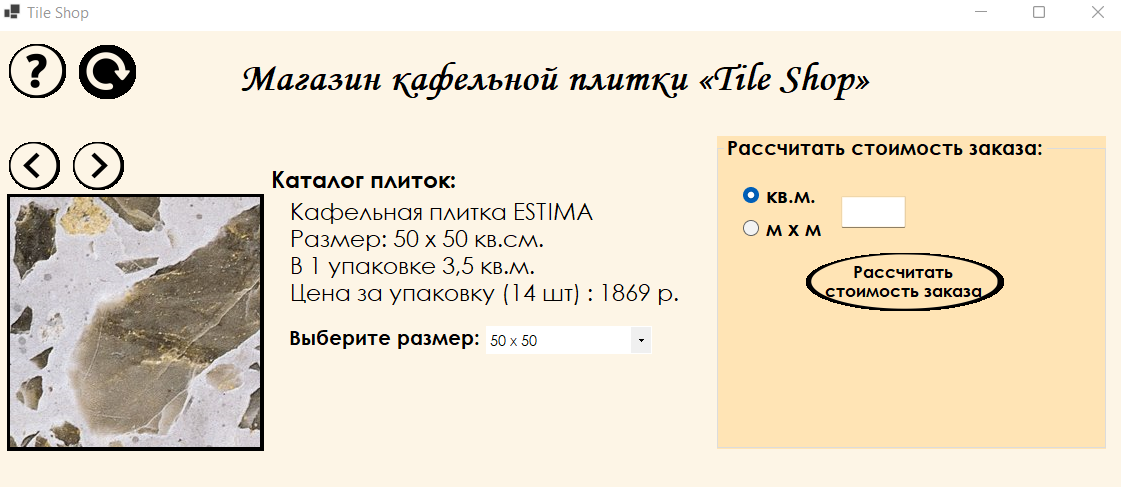
**Сначала:**

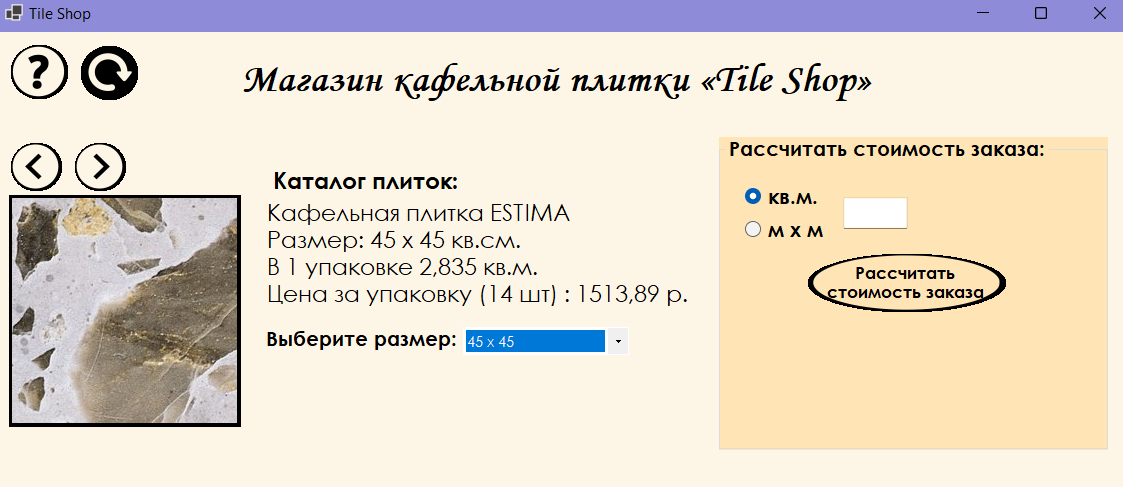


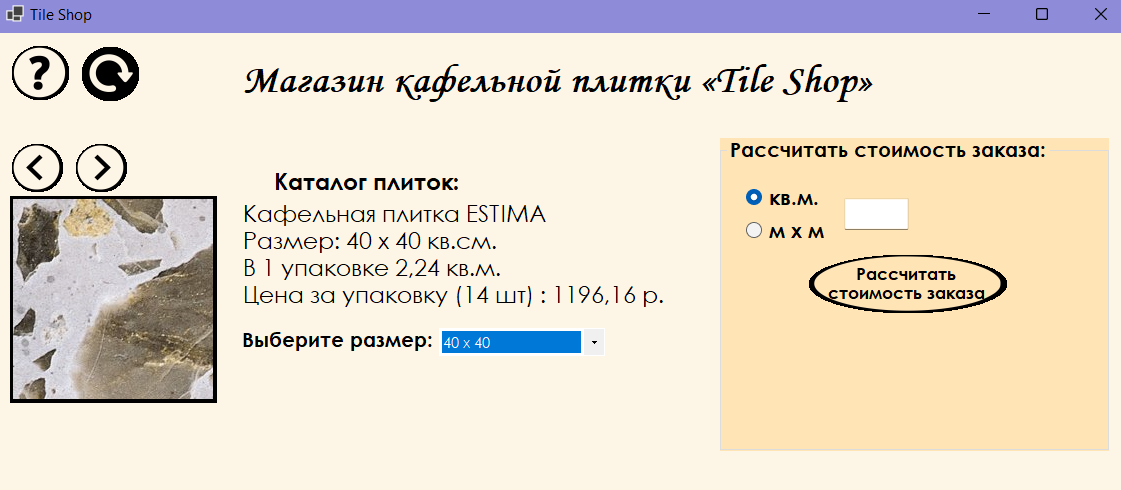
**С конца:**

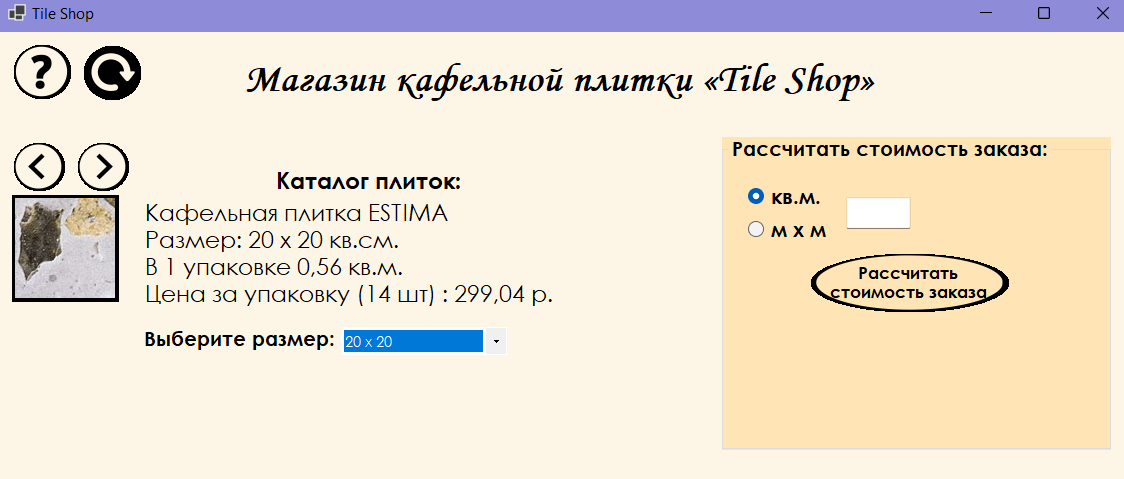


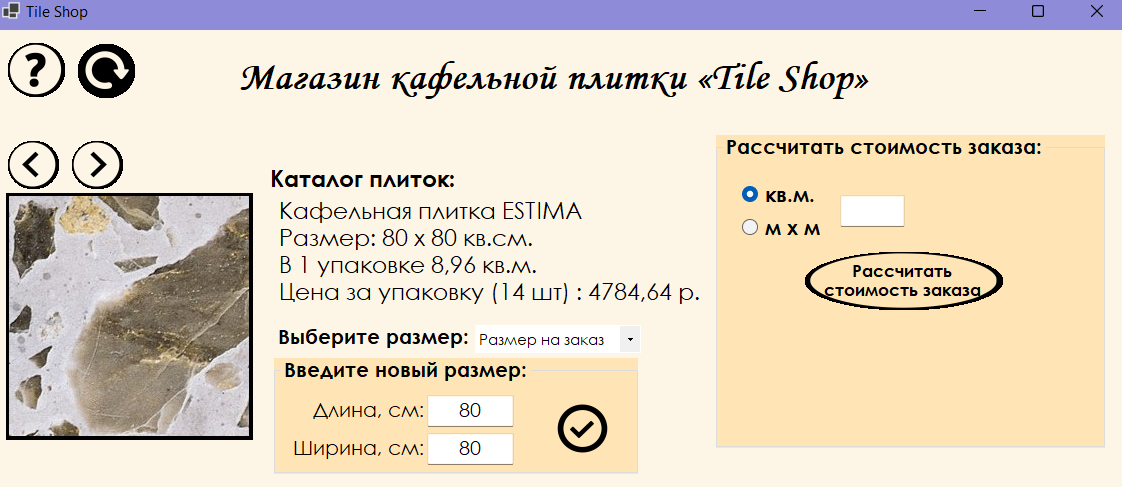
**Выбор размера:**

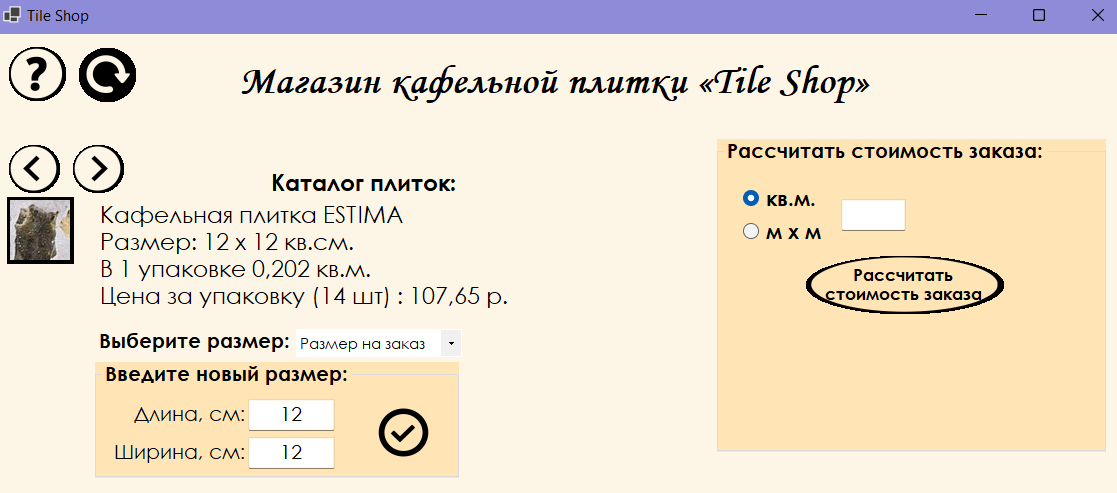












**Расчёт упаковок:**



