**Практическая работа №6**

**Перегрузка методов. Определение операций в классе**

**Цель:** Формирование навыков программирования C# и работы с системой MS Visual Studio,создания перегруженных методов.

**Теоретические сведения:**

**Перегрузкой** методов *(overloading)* называется использование методов с одним и тем же именем, но различным количеством и типами параметров. Компилятор по типу фактических параметров сам определяет, какой именно метод требуется вызвать. Это называется **разрешением** (*resolution*) перегрузки. Перегрузка методов – одна из простейших реализаций полиморфизма. Широко используется также перегрузка конструкторов.

Большинство стандартных операций тоже можно переопределять, что позволяет использовать объекты своих типов в составе выражений аналогично пе- ременным стандартных типов.

**Порядок выполнения работы:**

**Пример 1**

*Использование перегрузки методов. Вычисление периметров разных фигур.*

1. Создадим проект **сon6\_1\_ФИО**.
2. В классе **Program** создадим три статических метода с одинаковым именем **Perim**, но разными сигнатурами. В методе **Main** будем вызывать эти методы.

Изображение выглядит как текст

Автоматически созданное описание

1. Протестируем программу, вызывая метод **Perim** с разными параметрами. Обратим внимание на вариант метода с переменным числом параметров.

**Пример 2**

*Заказ номеров в гостинице. Использование перегрузки конструкторов.*

1. Создадим проект **сon6\_2\_ФИО**.
2. В едином пространстве имен **con6\_2\_ФИО** с классом **Program** создадим класс **Zakaz** с полями **fam** (фамилия), **size** (количество мест в номере), **comfort** (комфортность), методом **Show** (показать заказ) и четырьмя конструкторами с разным числом и типами параметров.

Изображение выглядит как текст

Автоматически созданное описание

1. В методе **Main** класса **Program** будем создавать объекты класса **Zakaz**

с разными параметрами и вызывать один и тот же метод **Show**.

**Изображение выглядит как текст

Автоматически созданное описание**

1. Протестируем программу, вызывая метод **Zakaz** с разными параметрами.

**Задания для самостоятельной работы**

1. Модифицируйте программу примера 1 (**сon6\_1\_ФИО**), добавив вариант перегрузки метода **Perim** для определения периметра квадрата по его стороне: **4\*a**.
2. Модифицируйте программу примера 2 (**сon6\_2\_ФИО**), добавив возможность многократного ввода заказа с клавиатуры (фамилия, количество мест в номере, комфорт). Завершение ввода – символ **Q**.

Создайте приложения, в которых определяются классы, поля, конструкторы, методы, создаются и инициализируются 2–3 объекта.

1. Класс **Figura.** Метод **ShowArea** перегружен. В зависимости от количества введенных параметров выводится название фигуры (один параметр – квадрат, два – прямоугольник, три – трапеция), вычисляется и выводится площадь.
2. Класс **Tour**. Метод **TourCalc** перегружен. Стоимость тура вычисляется в зависимости от количества и типа введенных параметров: без параметров – Минское море, бесплатно; один параметр (страна) – 1 день, 900 руб; два параметра (страна, количество дней **n**) – стоимость = 900\***n**.

**Практическая работа № 7**

**Инкапсуляция. Сокрытие полей, создание свойств**

Цель: Формирование навыков программирования C# и работы с системой MS Visual Studio, сокрытие полей классов.

**Теоретические сведения:**

Для защиты от нежелательных вмешательств доступ к полям и методам классов приходится закрывать или ограничивать (например, с помощью модификаторов private, protected). Для организации управления доступом к полям класса служат **свойства** (properties). Как правило, свойство определяет методы доступа к закрытому полю и имеет следующий синтаксис:

[ модификаторы] **тип Имя**

**{**

**{get** код\_доступа**}** // получение значения

**{set** код\_доступа**}** // установка значения

**}**

В C# свойство имеет то же имя, что и соответствующее скрытое поле, только первая буква заглавная, например: **поле** private int **age**, свойство public int **Age.**

При обращении к свойству автоматически вызываются указанные в нем блоки чтения **get** и установки **set**. Может отсутствовать либо часть get, либо set, но не обе одновременно. Если отсутствует set, свойство доступно только для чтения (read only), если отсутствует get – только для записи (write only).

**Пример 3**

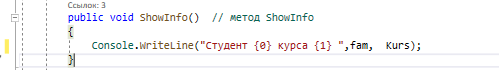
*Сокрытие полей, создание свойств.*

1. Создадим проект **сon6\_3\_ФИО**.
2. В едином пространстве имен **con6\_3\_ФИО** с шаблоном класса **Program** создадим класс **Student**. В этом классе объявим два поля **fam** (фамилия) и **kurs** (курс). Создадим конструкторы и метод **ShowInfo()**, который выводит информацию о студенте.

*Внимание!* Для удобства отладки программы все поля и методы сначала объявляем **public** (общедоступные). Имена полей вводим с маленькой буквы!

Изображение выглядит как текст

Автоматически созданное описание



Первоначальная общедоступность полей и методов класса **Student** позволяет при отладке программы в методе **Main()** класса **Program** создавать объекты класса **Student**, (т. е. описывать конкретных студентов по шаблону класса **Student**), а также вызывать метод **ShowInfo()**.

**class Program**

**{** static void Main()

**{** Student st1 = new Student("Иванов", 3); st1.ShowInfo(); Student st2 = new Student("", ‐7); st2.ShowInfo(); Console.ReadKey();

**}**

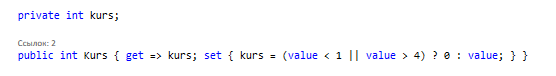
**}**

1. Протестируем программу с разными параметрами. Существенный недостаток – незащищенность от ввода абсурдных данных (например, можно ввести, что студент st2 не имеет фамилии и учится на отрицательном курсе –7.
2. **Инкапсулируем** данные (скроем и защитим поля), создав свойства с методами **set** и **get** для управления доступом к полям. Система Visual Studio позволяет автоматизировать этот процесс.
3. Для этого устанавливаем курсор на **имя\_поля** (например, **fam**), в меню **Рефакторинг** (Refactoring) выбираем пункт **Инкапсулировать поле** (Incapsulate field). Доступ к полю **fam** будет изменен на **private** (закрытый) и методом **Fam** сгенерирован шаблон общедоступного **свойства** с именем **Fam** (с большой буквы), которое и будет использоваться теперь вместо поля **fam**.
4. Создадим свои правила доступа. Для этого будем вводить необходимый код в автоматически сгенерированные шаблоны **set** и **get**. Например, все фамилии будем хранить большими буквами (преобразование зададим в блоке **set**). А если фамилия не введена (поле **fam** пустое), то блок **get** будет возвращать значение «неизвестный». Измененный фрагмент кода будет выглядеть так:

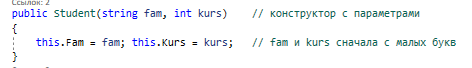
Изображение выглядит как текст

Автоматически созданное описание

1. Аналогичным образом инкапсулируем поле **kurs**. Защитим его от ввода и хранения абсурдных значений. Например, при вводе чисел <1 или >4 в поле **kurs** будет сохранено значение 0 (поступай на подготовительный курс!).

****

1. На завершающем этапе заменим в конструкторе **имена скрываемых полей** (с малой буквы) на **имена открытых свойств** (с большой буквы). Проверьте это должно произойти автоматически:



1. Протестируем окончательный вариант с разными параметрами.

**Задания для самостоятельной работы**

1. Модифицируйте программу **сon6\_3\_ФИО**.
2. Добавьте еще два поля: имя **name** и возраст **age**. Инкапсулируйте их, введя ограничения на возраст от 15 до 35 лет.
3. Добавьте конструктор с четырьмя параметрами:

**public Student(string fam, string name, int kurs, int age)**.

1. Протестируйте программу, изменяя параметры инициализации, например;

Student st3 = new Student("Петров", "Петр", ‐7, 120);

st3.ShowInfo();

1. Создайте приложения, в которых определяются классы, поля, конструкторы, создаются и инициализируются 2–3 объекта. Поля инкапсулируются. Информация выводится методом **Show.**
2. Создается класс **Avto** с полями: марка автомобиля **brand**, цвет **color**, скорость **skor**. Поля инкапсулируются с ограничениями (скорость от 20 до 120 км/ч).
3. Создается класс **Kadry** с полями: фамилия **fam**, возраст **age**, должность **dol**, стаж **staj**. Поля инкапсулируются с ограничениями (возраст от 16 до 60, стаж от 0 до 45).
4. Создается класс **Computer** с полями: модель **model,** объем ОЗУ **ram** и жесткого диска **hdd**. Поля инкапсулируются с ограничениями (объем ОЗУ от 2 до 32 Гбайт, жесткого диска от 200 до 2000 Гбайт.
5. \*Создается класс **Tovar** с полями: название **name**, цена **price**, количество **kvo**. Поля инкапсулируются с ограничениями (цена от 1 до 20, количество от 0 до 10). Вычисляется стоимость заказанного товара каждого вида и всего заказа.