Лабораторная работа №7. Обобщенное программирование, шаблоны

Практические задания

На оценку «**удовлетворительно**» необходимо выполнить часть обязательных заданий и набрать не менее 4**0 баллов**.

На оценку «**хорошо**» необходимо выполнить все обязательные и часть дополнительных заданий, набрав не менее **220 баллов**.

На оценку «**отлично**» необходимо выполнить все обязательные и часть дополнительных заданий, набрав не менее **500 баллов**.

Обязательные задания

Задание 1

Вариант 1. 15 баллов

Разработайте шаблонную функцию **Sort2**, выполняющую упорядочивание своих аргументов, переданных ей по ссылке, в порядке возрастания:

```
template <typename T>
void Sort2(T & first, T & second)
```

Разработайте специализированную версию данной шаблонной функции, выполняющую упорядочивание двух аргументов типа const char в лексикографическом порядке:

```
template <>
void Sort2<const char*>(const char* & first, const char* & second)
```

Продемонстрируйте работу данной функции на примере программы, выполняющей упорядочивание двух строк, двух чисел с плавающей запятой, **трех** целых чисел, а также двух указателей типа **const char***. Входные данные программы поступают со стандартного потока ввода[1].

Проиллюстрировать работу программы по упорядочиванию указателей типа **const char*** можно, например, так:

```
string s1, s2;

// Считываем строки в переменные s1 и s2
getline(s1);
getline(s2);

// Получаем указатели на хранящиеся в строках массивы символов
const char* p1 = s1.c_str();
const char* p2 = s2.c_str();

// Упорядочиваем указатели в лексикографическом порядке
Sort2(p1, p2);

// Выводим результат output
cout << p1 << "\n" << p2 << "\n";
```

Вариант 2. 30 баллов

Разработайте шаблонную функцию **FindMax**, выполняющую поиск максимального элемента в массиве. В случае, если такой элемент присутствует (массив не пустой), то значение максимального элемента должно быть занесено в аргумент **maxValue**, передаваемый по ссылке, а сама функция вернуть значение «истина». Если массив был пустым, функция должна вернуть значение «ложь».

Прототип функции FindMax:

```
template < typename T >
bool FindMax(std::vector<T> const& arr, T& maxValue)
```

Продемонстрируйте работу данной функции для поиска максимального элемента в массиве целых чисел, массиве чисел с плавающей запятой, а также элементов типа std::string.

Разработайте специализированную версию данной функции, выполняющую поиск максимального элемента в массиве const char*, интерпретируемых как строки символов, а не обычные указатели. Продемонстрируйте работу данной функции для нахождения максимального элемента в массиве указателей на строки.

```
template <>
bool FindMax<const char *>(std::vector<const char*> const& arr, T& maxValue)
```

Внимание, необходимо обосновать предоставление Вашей реализацией функции FindMax гарантий строгой безопасности исключений, а также поддержи семантики выполнения «commit-or-rollback».

Вариант 3. 50 баллов

Разработайте функцию **FindMaxEx**, являющуюся расширенной версией функции FindMax и позволяющую задавать критерий сравнения элементов при помощи дополнительного параметра less:

```
template < typename T, typename Less>
bool FindMax(std::vector<T> const& arr, T& maxValue, Less const& less)
```

[1] Не забудьте вывести пояснение для пользователя, например «Enter two integers:».

Предикат Less – функция или функтор, принимающий 2 аргумента типа T и возвращающая true, если левый аргумент предшествует правому, в противном случае - false. Данный предикат задает критерий сравнения элементов, используемый для поиска максимального элемента.

Продемонстрируйте работу данной функции для поиска спортсмена (ФИО, рост, вес) в массиве других спортсменов, обладающего максимальным ростом, а также спортсмена с максимальным весом.

Внимание, необходимо обосновать предоставление Вашей реализацией функции FindMax гарантий строгой безопасности исключений, а также поддержи семантики выполнения «commit-or-rollback».

Вариант 4. 30 баллов.

Разработайте шаблонную функцию **ArraySum**, вычисляющую сумму элементов массива.

```
template <typename T>
T ArraySum(std::vector<T> const& arr)
```

Для пустого массива должно возвращаться значение конструктора по умолчанию для типа T.

Продемонстрировать работу данной функции для нахождения суммы элементов массив целых чисел, массива чисел с плавающей запятой, а также для массива строк (в этом случае должна выполняться конкатенация строк).

Внимание, необходимо обосновать предоставление Вашей реализацией функции ArraySum гарантий строгой безопасности исключений, а также поддержи семантики выполнения «commit-or-rollback».

Задание 2

Вариант 1 – 100 баллов

Разработайте шаблонный класс CMyArray, представляющий собой массив элементов некоторого типа T. Массив должен предоставлять следующие возможности:

- Возможность добавления элемента в конец массива
- Возможность получения количества элементов, содержащихся в массиве
- Возможность осуществления индексированного доступа к элементам массива при помощи оператора «[]». В случае, если индекс элемента выходит за пределы массива, должно выбрасываться исключение std::out_of_range
- Возможность изменения длины массива при помощи метода Resize(). В случае, если новая длина массива больше прежней, вставляемые в конец массива элементы должны инициализироваться значением по умолчанию для типа T.
- Возможность опустошения массива (удаления всех его элементов) при помощи метода Clear.
- Конструктор копирования и оператор присваивания

Все методы класса должны CMyArray должны предоставлять строгую гарантию безопасности исключений и семантику «commit-or-rollback», а деструктор – гарантировать отсутствие исключений. **Массив должен быть реализован своими силами без использования класса std::vector и ему подобных**.

Программа должна демонстрировать работу шаблонного класса CMyArray в качестве массива строк (std::string) и массива чисел с плавающей запятой.

Бонус в 20 баллов

Дополнительно можно получить до 20 баллов, если реализовать в классе CMyArray шаблонный оператор присваивания, выполняющий присваивание элементов массива одного типа массиву элементов другого

типа с использованием статического преобразования типов (**static_cast**). Данный оператор также должен быть безопасен к возникновению исключений.

Вариант 2 – 80 баллов

Разработайте шаблонный класс CMyStack, представляющий собой стек элементов некоторого типа T. Массив должен предоставлять следующие возможности:

- Вталкивание элемента на вершину стека
- Выталкивание элемента с вершины стека
- Возвращение элемента с вершины стека
- Информацию о том, пуст ли стек.
- Возможность опустошения стека (удаления всех его элементов) при помощи метода Clear.
- Конструктор копирования и оператор присваивания

Все методы класса должны CMyStack должны предоставлять строгую гарантию безопасности исключений и семантику «commit-or-rollback», а деструктор – гарантировать отсутствие исключений.

Программа должна демонстрировать работу стека целых чисел, а также стека строк.

Вариант 3 – 300 баллов

Разработайте шаблонный класс CMySet, реализующий множество элементов некоторого типа, а также операции над данным множеством. Самостоятельно реализуйте соответствующие структуры данных, необходимые для хранения элементов множества.

Все элементы в составе множества являются уникальными, т.е. множество не содержит двух одинаковых элементов.

Порядок следования элементов в составе множеств не имеет значения.

Продемонстрируйте использование разработанного класса множеств для решения следующей задачи:

На потоке учатся 3 группы студентов, но не менее 15 студентов в каждой группы. С каждой группы по 5 студентов приняли участие в олимпиаде, из которых 3 студента заняли призовые места (выбираются случайным образом). Необходимо вывести с каждой группы тех студентов, которые не заняли призовых мест и вывести их имена в output.

Все методы класса CMySet должны обеспечивать строгую гарантию безопасности исключений и поддерживать семантику «commit-or-rollback»

Класс должен реализовывать следующие **операции над множествами**:

Объединение множеств (Union)

Результатом объединения двух множеств является множество, содержащее элементы, входящие в состав этих множеств.

$$\{a, b, c, d\} \cup \{c, d, e, f\} = \{a, b, c, d, e, f\}$$

Пересечение множеств (Intersection)

Результатом пересечения двух множеств является множество, содержащее элементы, которые ходят одновременно в состав обоих множеств

$$\{a, b, c, d\} \cap \{c, d, e, f\} = \{c, d\}$$

Разность множеств (Difference)

Разность двух множеств – множество, содержащее в себе только те элементы первого множества, которые отсутствуют во втором множестве.

$$\{a, b, c, d\} - \{c, d\} = \{a, b\}$$

Симметричная (симметрическая) разность множеств (Symmetric difference)

Симметрическая разность двух множеств – множество элементов данных множеств, которое принадлежит только одному из них.

$$\{a, b, c, d\}\Delta\{a, b, c, e\} = \{d, e\}$$

Принадлежность элемента множеству (Contains)

Определяет, содержит ли множество заданный элемент

Принадлежность одного множества другому (ConstainsSubset)

Определяет, содержит ли одно множество в себе другое. Любое множество всегда содержит в себе пустое множество.

Определение пустоты множества

Определяет, является ли множество пустым, т.е. не содержит ни одного элемента.

Определение равенства двух множеств

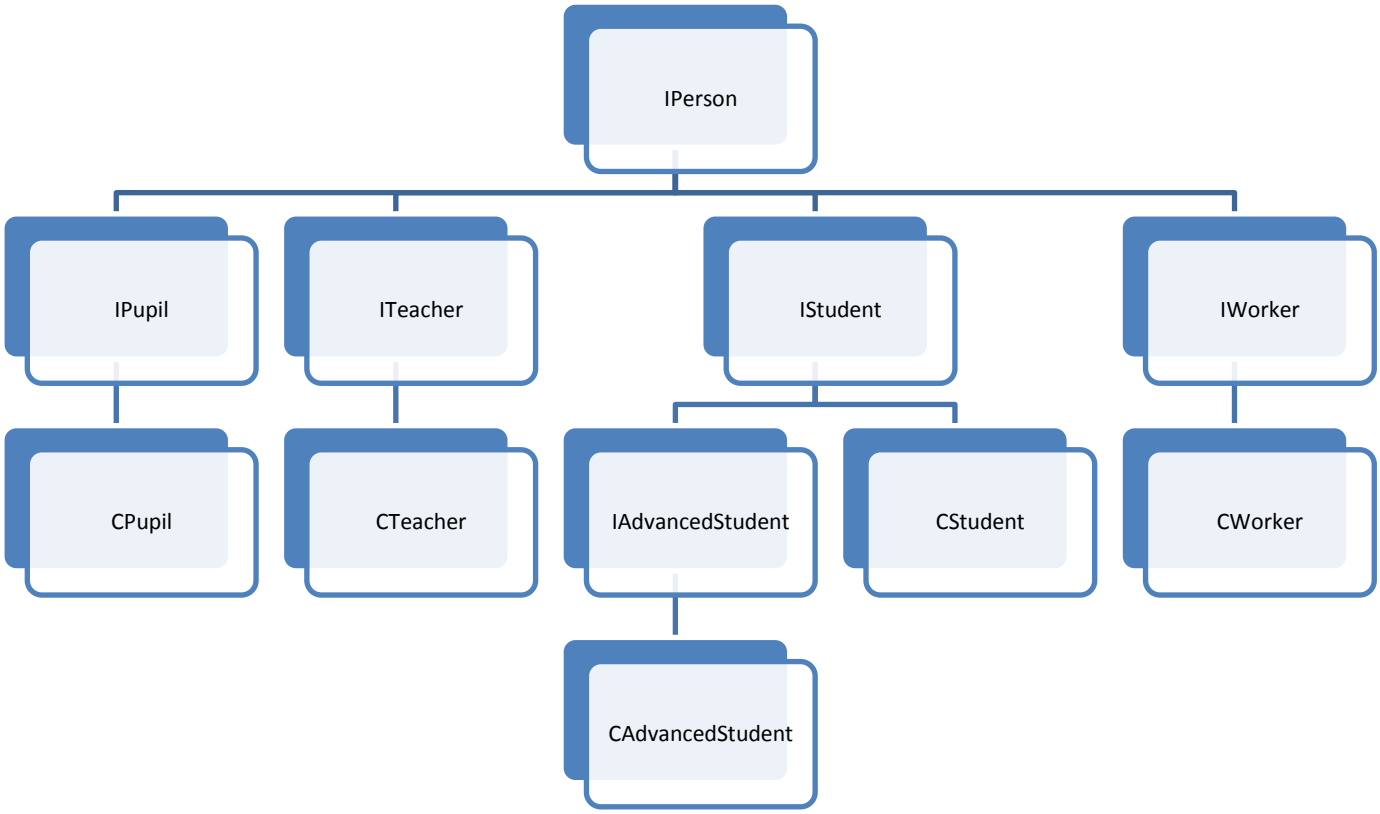
Определяет, равно ли одно множество другому, т.е. каждое из множеств содержит все элементы другого множества. Два пустых множества считаются равными.

Дополнительные задания

Задание 3

Вариант 1. 150 баллов

Имеется следующая иерархия классов и интерфейсов, моделирующих иерархию объектов в обществе (человек, учитель, ученик, студент, рабочий, аспирант):



Свойства, которыми обладают сущности данной иерархии:

Сущность	Свойства
Человек	Имя Фамилия Отчество Адрес
Ученик	Название школы Класс, в котором учится (например, «7б»)
Преподаватель	Название предмета, который ведет преподаватель
Студент	Название ВУЗ-а Номер студенческого билета
Аспирант	Тема диссертации
Рабочий	Специальность

Как видно по иерархической диаграмме, все конкретные классы-наследники интерфейса IPerson должны однотипным образом реализовать его свойства «Фамилия», «Имя», «Отчество» и «Адрес». Кроме того, аспирант и студент также имеют общую реализацию свойств «Название ВУЗа» и «Номер студенческого билета».

Чтобы избежать дублирования кода данных свойств перечисленных сущностей иерархии необходимо разработать шаблонные реализации интерфейсов IStudent (CStudentImpl) и IPerson(CPersonImpl):

```
template <typename Base>
class CPersonImpl : public Base
{
	//...
};

template <typename Base>
class CStudentImpl : public CPersonImpl<Base>
{
	//...
};
```

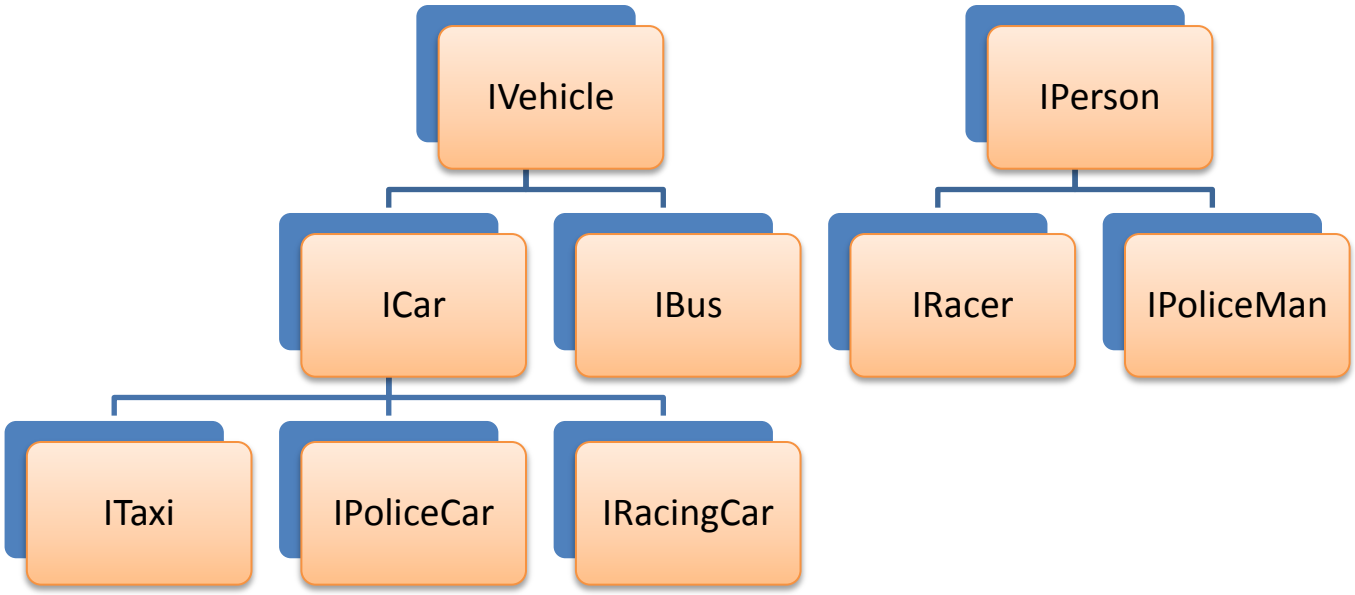
С использованием данных реализаций строятся прочие классы, например:

```
class CWorker : public CPersonImpl<IPerson
{
	//...
}

class CStudent : public CStudentImpl<IStudent>
{
	//...
};

class CAdvancedStudent : public CStudentImpl<IAdvancedStudent>
{
	//...
};
```

Вариант 2. 200 баллов

Различные виды транспорта могут перевозить различные типы пассажиров – автобусы и такси – всех людей, полицейские автомобили – полицейских, гоночные машины – гонщиков.

Имеется следующая изначальная иерархия типов транспортных средств и пассажиров:




```
// человек
class IPerson
{
public:
	// имя
	virtual std::string GetName()const = 0;
};

// полицейский
class IPoliceMan : public IPerson
{
public:
	// название полицейского департамента
	virtual std::string GetDepartmentName()const = 0;
};

// гонщик
class IRacer : public IPerson
{
public:
	// количество наград
	virtual size_t GetAwardsCount()const = 0;
};
```

Нам нужен механизм, позволяющий размещать в определенных видах транспорта соответствующие типы пассажиров. Например, в такси или автобус мы должны иметь возможность разместить любых людей (включая гонщиков и полицейских), в полицейскую машину – только полицейских, а в гоночную – только гонщиков.

Можно было бы спроектировать следующие классы транспортных средств:

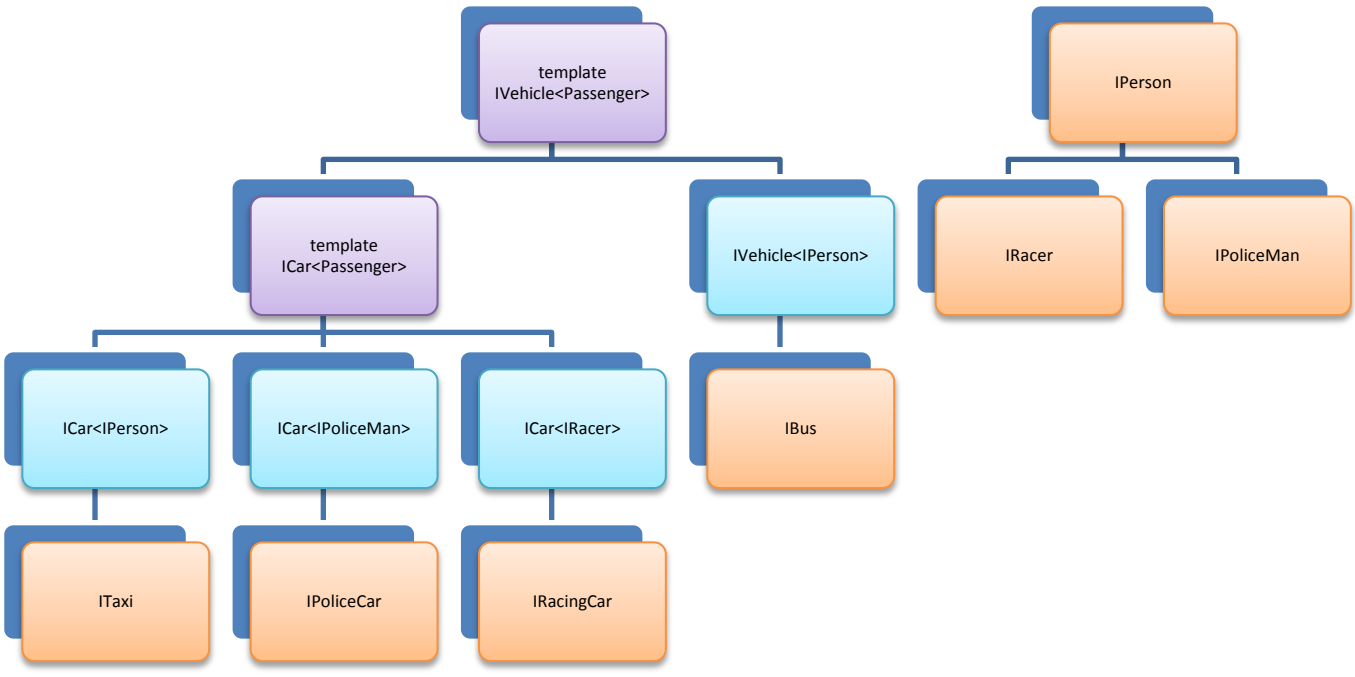
```
class IVehicle
{
};

class ICar : public IVehicle
{
};

class IRacingCar : public ICar
{
public:
	virtual void AddRacer(boost::shared_ptr<IRacer> pRacer) = 0;
```

```
};

class IPoliceCar : public ICar
{
public:
	virtual void AddPoliceMan(boost::shared_ptr<IPoliceMan> pPoliceMan) = 0;
};

class ITaxi : public ICar
{
public:
	virtual void AddPassenger(boost::shared_ptr<IPerson> pPerson) = 0;
};

class IBus : public IVehicle
{
public:
	virtual void AddPassenger(boost::shared_ptr<IPerson> pPerson) = 0;
};
```

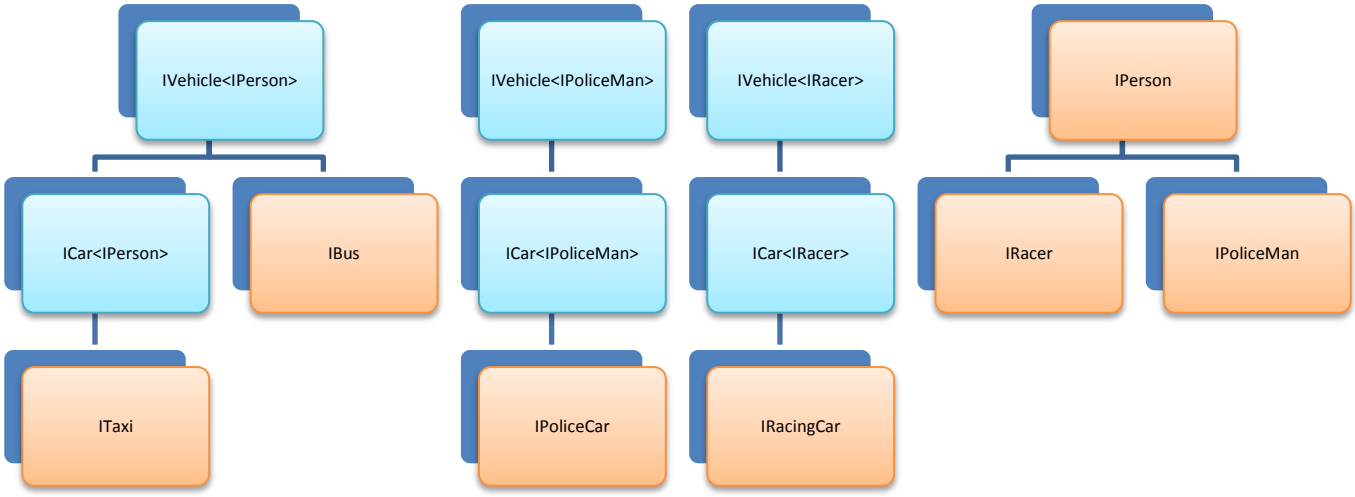
Очевидно, что такое решение приведет нас к тому, реализация методов посадки нужного типа человека в транспортное средство будет во всех классах одинаковой. Поэтому лучше сделать интерфейсы IVehicle и ICar шаблонными, где в качестве параметра шаблона будет выступать тип людей, которые возможно на данном типе транспорта перевозить.

Интерфейсы «такси», «полицейский автомобиль», «гоночный автомобиль» и «автобус» будут образованы при помощи наследования от шаблонных классов «автомобиль» и «транспорт».

Обновленная иерархия:



Фактически во время компиляции на этапе инстанциирования шаблонов компилятором будет построена следующая иерархия (произойдет «расщепление» базового шаблонного класса IVehicle):



Исходный код получившейся иерархии (В класс IVehicle были добавлены несколько методов):

```
template <typename Passenger>
class IVehicle
{
public:
	// сигнализирует о том, пусто ли транспортное средство
	virtual bool IsEmpty()const = 0;

	// сигнализирует о том заполнено ли транспортное средство полностью
	virtual bool IsFull()const = 0;

	// возвращает общее количество мест
	virtual size_t GetPlaceCount()const = 0;

	// возвращает количество пассажиров на борту
	virtual size t GetPassengerCount()const = 0;

	// высаживает всех пассажиров
	virtual void RemoveAllPassengers()const = 0;

	// добавить пассажира на борт
	// т.к. пассажир может быть полиморфным типом, принимаем его по умному указателю
	// Если нет места, выбрасывается ислючение std::logic_error
	virtual void AddPassenger(boost::shared_ptr<Passenger> pPassenger) = 0;

	// Получить ссылку на пассажира с заданным индексом
	// выбрасывает исключение std::out_of_range в случае недопустимого индекса
	virtual Passenger const& GetPassenger(size_t index)const = 0;

	// убрать пассажира с заданным индексом
	// выбрасывает исключение std::out_of_range в случае недопустимого индекса
	virtual void RemovePassenger(size_t index)const = 0;
};

template <typename Passenger>
class ICar : public IVehicle<Passenger>
{
};

class IRacingCar : public ICar<IRacer>
{
public:
};

class IPoliceCar : public ICar<IPoliceMan>
{
```

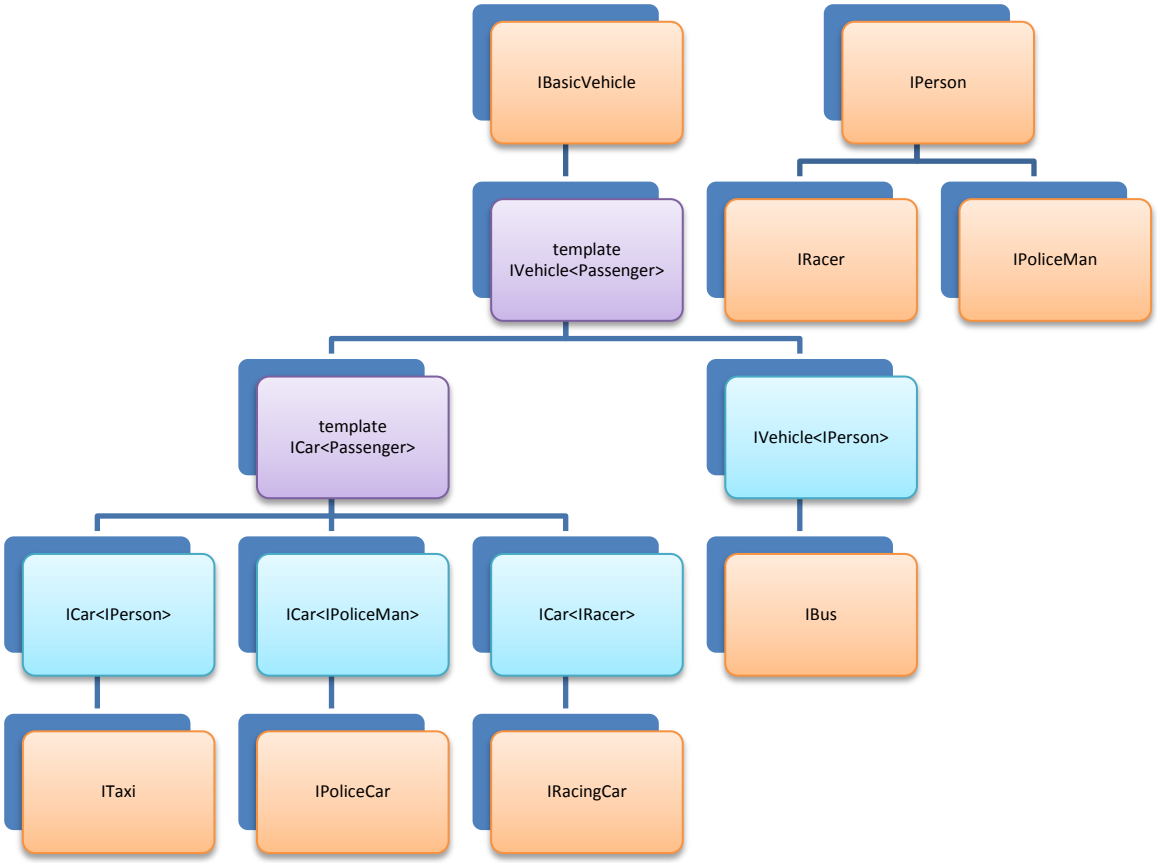
```
public:
};

class ITaxi : public ICar<IPerson>
{
public:
};

class IBus : public IVehicle<IPerson>
{
public:
};
```

Проанализировав класс IVehicle мы приходим к выводу, что часть методов данного класса не зависит от типа пассажиров:

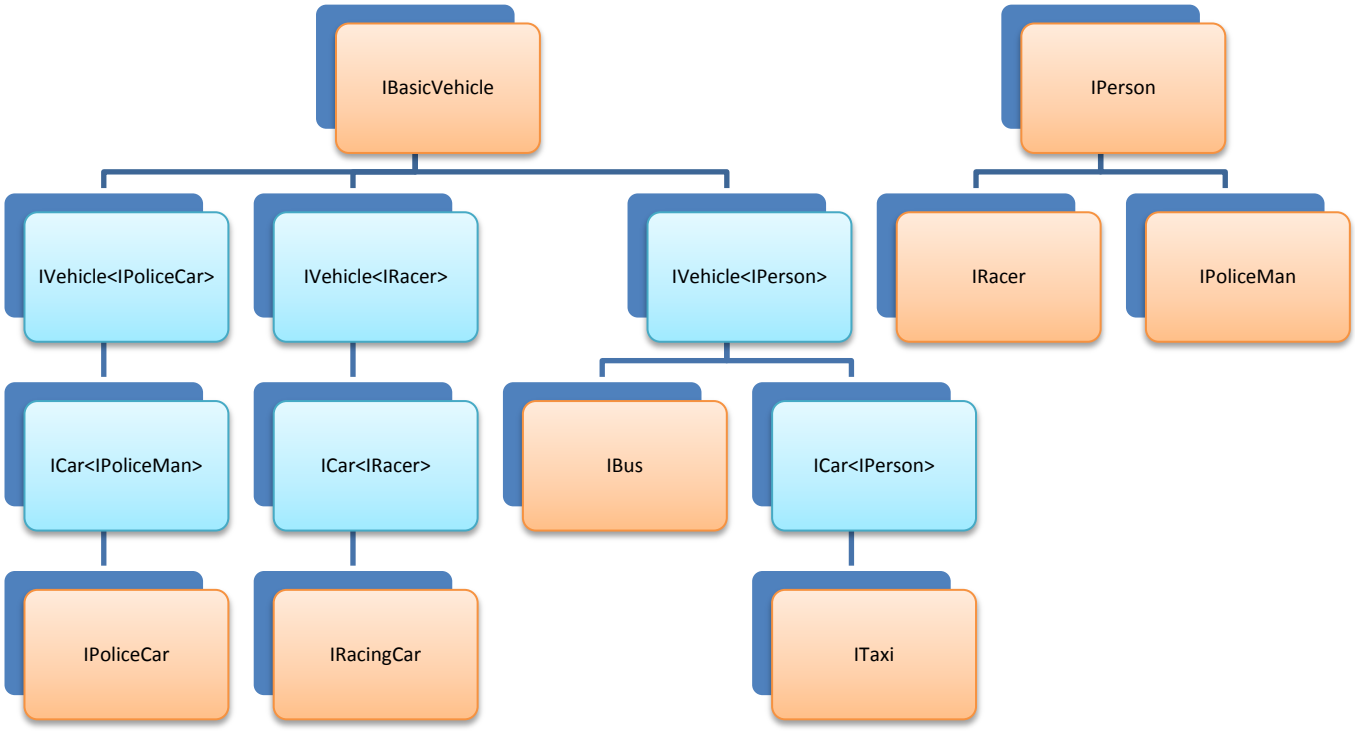
```
    // сигнализирует о том, пусто ли транспортное средство
    virtual bool IsEmpty()const = 0;

    // сигнализирует о том заполнено ли транспортное средство полностью
    virtual bool IsFull()const = 0;

    // возвращает общее количество мест
    virtual size_t GetPlaceCount()const = 0;

    // возвращает количество пассажиров на борту
    virtual size_t GetPassengerCount()const = 0;

    // высаживает всех пассажиров
    virtual void RemoveAllPassengers()const = 0;
```

Данные методы можно выделить в базовый **нешаблонный** интерфейс **IBasicVehicle**, не зависящий от типа пассажиров.

Обновленная иерархия будет выглядеть следующим образом:



На этапе инстанциирования шаблонов из нее будет построена следующая:



Исходный код обновленной иерархии интерфейсов (в классы ряда транспортных средств добавлены методы, специфичные для данного типа транспортных средств):

```
// базовое транспортное средство
class IBasicVehicle
{
public:
```

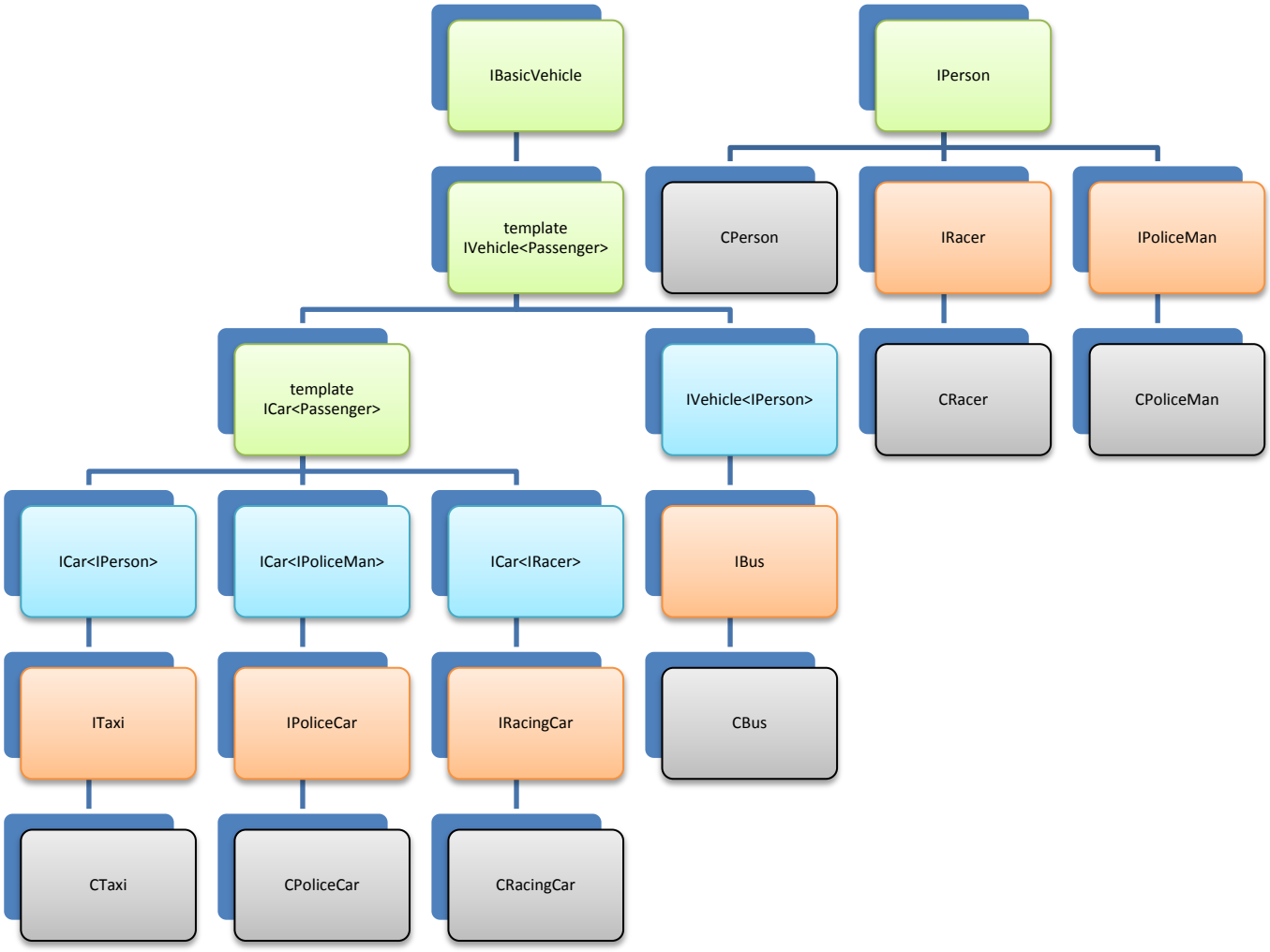
```
    // сигнализирует о том, пусто ли транспортное средство
    virtual bool IsEmpty()const = 0;

    // сигнализирует о том заполнено ли транспортное средство полностью
    virtual bool IsFull()const = 0;

    // возвращает общее количество мест
    virtual size_t GetPlaceCount()const = 0;

    // возвращает количество пассажиров на борту
    virtual size_t GetPassengerCount()const = 0;

    // высаживает всех пассажиров
    virtual void RemoveAllPassengers()const = 0;
};

// транспортное средство предназначенное для провозки заданного типа пассажиров
template <typename Passenger>
class IVehicle : public IBasicVehicle
{
public:
    // добавить пассажира на борт
    // т.к. пассажир может быть полиморфным типом, принимаем его по умному указателю
    // Если нет места, выбрасывается ислючение std::logic_error
    virtual void AddPassenger(boost::shared_ptr<Passenger> pPassenger) = 0;

    // Получить ссылку на пассажира с заданным индексом
    // выбрасывает исключение std::out_of_range в случае недопустимого индекса
    virtual Passenger const& GetPassenger(size_t index)const = 0;

    // убрать пассажира с заданным индексом
    // выбрасывает исключение std::out_of_range в случае недопустимого индекса
    virtual void RemovePassenger(size_t index)const = 0;
};

// марка автомобиля
enum MakeOfTheCar
{
    BMW,
    MITSUBISHI,
    FORD,
    MERCEDES,
    TOYOTA,
    KIA,
    FERRARI,
    PORSCHE,
    LEXUS,
    NISSAN,
    INIFINITI,
};

// автомобиль, перевозящий заданный тип пассажиров
template <typename Passenger>
class ICar : public IVehicle<Passenger>
{
public:
    virtual MakeOfTheCar GetMakeOfTheCar()const = 0;
};

// марка автобуса
class IBus : public IVehicle<IPerson>
{
};

class IPoliceCar : public ICar<IPoliceMan>
{
```

```
};

class IRacingCar : public ICar<IRacer>
{
};

class ITaxi : public ICar<IPerson>
{
};
```

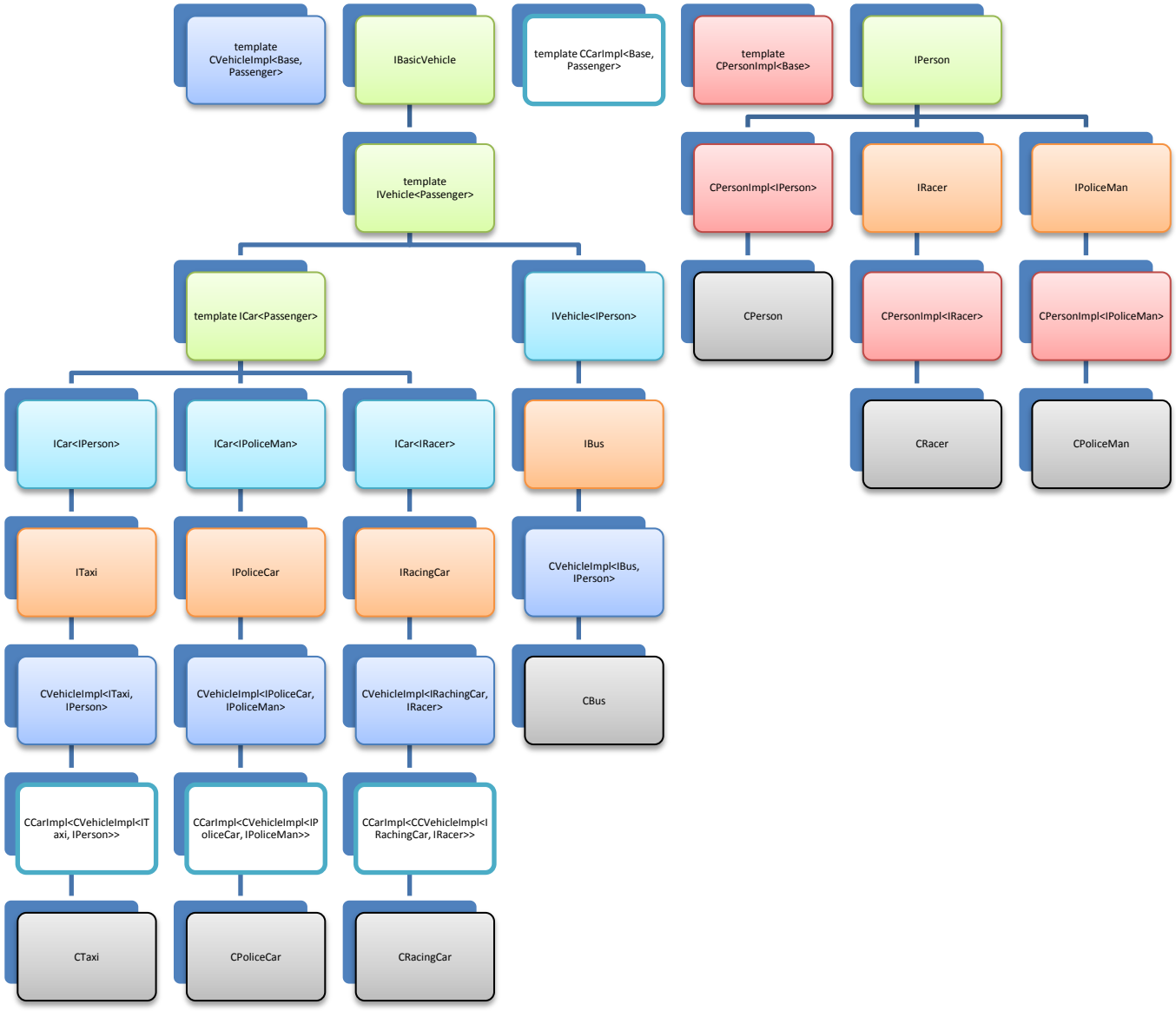
Покончив с проектированием интерфейсов, настало время подумать о деталях реализации. Нам понадобятся следующие классы, реализующие следующие сущности предметной области

- Люди
  - CPerson – обычный человек
  - CRacer - гонщик
  - CPoliceMan - полицейский
- Транспорт
  - CTaxi - такси
  - CPoliceCar – полицейская машина
  - CRacingCar – гоночная машина
  - CBus - автобус

Получится следующая иерархия сущностей предметной области:



При реализации большинства методов данных сущностей мы неизбежно при текущей иерархии столкнемся с дублированием кода – нам необходимо будет реализовать в классах людей методы класса CPerson. Для устранения дублирования кода необходимо разработать шаблонные реализации сущностей «автомобиль», «транспортное средство», «человек» (на диаграмме выделены зеленым цветом):



Не смущайтесь визуально разросшейся иерархии классов (большинство из них – конкретизированные версии одних и тех же шаблонов – выделены одинаковым цветом). Фактически, реальных классов будет всего несколько:

- Шаблонные реализации интерфейсов
  - CVehicleImpl – шаблонная реализация методов транспортного средства
  - CCarImpl – шаблонная реализация методов, добавленных в интерфейс «Автомобиль»
  - CPersonImpl – шаблонная реализация методов человека
- Конкретные классы сущностей предметной области
  - CTaxi – реализует методы интерфеса ITaxi (в нашем случае таких методов нет), наследуется от шаблона CCarImpl
  - CPoliceCar – реализует методы интерфейса IPoliceCar (в нашем случае таких методов нет), наследуется от шаблона CCarImpl
  - CRacingCar – реализует методы интерфейс IRacingCar (в нашем случае таких методов также нет)

- CPerson – фактически, инстанция шаблона CPersonImpl,параметризованного интерфейсом IPerson
- CRacer – реализует методы интерфейса IRacer, наследуется от шаблона CPersonImpl, параметризованного интерфейсом IRacer
- CPoliceMan – реализует методы интерфейса IPoliceMan, наследуется от шаблона CPersonImpl, параметризованного интерфейсом IPoliceMan

Необходимо реализовать вышеописанные классы и смоделировать в программе с их помощью следующую историю (из начальных данных задачи необходимо выделить ключевые моменты и реализовывать только их):

- **Полицейский Джон Смит** из **Северо-западного полицейского участка** садится в свой служебный **пятиместный** полицейский автомобиль марки **Ford**. К нему в автомобиль садится его знакомый коп **Джим Кларк** из **Юго-восточного полицейского участка**. Программа должна вывести имена полицейских, находящихся в упомянутой полицейской машине, а также имена их полицейских участков.
- Упомянутые полицейские ссорятся. В сердцах **полицейский Джим Кларк выходит из упомянутой машины** своего теперь уже бывшего друга и останавливает **двухместное такси** марки **Тойота**, которым управляет выходец из Индии **Раджа Ганди**, везущий **гонщика Михаэля Шумахера** на стадион, на котором через 15 минут должен начаться чемпионат мира. **«Убедив» таксиста** при помощи своего табельного оружия **покинуть машину**, полицейский Джим **садится в такси** на место водителя и соглашается подвезти своего любимого гонщика до стадиона.
- Таксист **пытается вернуться** в свой автомобиль, но при попытке сесть в машину, **ловит исключение std::logic_error**, которое как бы говорит ему о нелогичности такого поступка, т.к. в машине нет свободных мест.

Вариант 3. 250 баллов

Необходимо разработать шаблонный класс **CFamily**, представляющий обобщенную реализацию сущности «семья живых существ», а также ряд вспомогательных шаблонных классов, используемый семьей в ходе работы.

Семья представляет собой пару особей одного вида, которые при желании и возможности могут еще и обзавестись потомством своего вида. Создание семьи особями одного пола не запрещается, но возможность заводить детей для них будет отсутствовать.

Первоначальный каркас шаблона CFamily выглядит следующим образом:

```
// Шаблон "семья" некоторых существ
template <
   typename Creature     // Вид особей, составляющих семью
>
class CFamily
{
public:
   // Конструктор, создающий семью из пары особей одного вида spouse1 и spouse2
   CFamily(Creature const& spouse1, Creature const& spouse2);

   // Возвращает первую особь семьи
   Creature GetSpouse1()const;

   // Возвращает вторую особь
   Creature GetSpouse2()const;

   // Возвращает особь-ребенка с заданным индексом, выбрасывая исключение
   // std::range_error, в случае выхода индекса за допустимые пределы
   Creature GetChild(size_t index)const;
```

```
  // Возвращает количество детей в семье
  size_t GetChildCount()const;

  /* Создание новой особи (ребенка) и добавление его в массив детей семьи
    При попытке вызова данного метода у гомосексуальной пары должно выбрасываться
    исключение std::logic_error
  */
  Creature MakeChild();
private:
  ...
};
```

Поскольку по условию задачи создание детей доступно только гетеросексуальным парам, метод MakeChild должен иметь возможность определить пол каждого из супругов. Поскольку разные классы несвязанных друг с другом особей могут в общем случае предоставлять несовместимые друг с другом средства по определению пола, шаблонному классу семьи потребуется передать дополнительный шаблонный параметр, который будет использоваться как функтор, инкапсулирующий определение пола особи и возвращающий его.

Для облегчения построения функторов определения пола, создадим шаблонный класс стандартного определителя пола **CStandardGenderDeterminer**, вызывающий метод GetGender() у особи[2].

```
// Функтор "Определитель пола", выполняющий стандартное определение пола
// некоторого существа, имеющего метод GetGender()
template <typename Creature>
class CStandardGenderDeterminer
{
public:
  Gender operator()(Creature const& creature)const
  {
    return creature.GetGender();
  }
};
```

Шаблон класса «Семья» претерпит следующие изменения:

```
// Шаблон "семья" некоторых существ
template <
  typename Creature,    // Вид особей, составляющих семью
  // Функтор, определяющий пол особи данного вида
  typename GenderDeterminer = CStandardGenderDeterminer<Creature>
>
class CFamily
{
public:
  …
  /* Создание новой особи (ребенка) и добавление его в массив детей семьи
    При попытке вызова данного метода у гомосексуальной пары должно выбрасываться
    исключение std::logic_error
  */
  Creature MakeChild()
  {
    // Создаем определитель пола
    GenderDeterminer genderDeterminer;

    // Определяем пол супругов при помощи определителя пола
    Gender gender1 = genderDeterminer(GetSpouse1());
```

[2] Предположим, что в нашей программе большинство классов особей будут предоставлять данный метод. Для классов особей, в которых для определения пола используется метод с другой сигнатурой, можно будет реализовать отдельный класс или шаблон.

```
      Gender gender2 = genderDeterminer(GetSpouse2());

      // Если пол супругов разный, создаем ребенка. Иначе выбрасываем исключение
      // о нелогичности вызова данного метода
      …
   }
   …
};
```

Процесс создания новой особи заслуживает особого рассмотрения. Особи разного вида могут конструироваться по-разному. Если для создания щенков и котят достаточно лишь задать пол детеныша, то появление ребенка в человеческой семье сопровождается еще и наделением ребенка именем, а также получением свидетельства о рождении.

Для решения данной проблемы в класс «Семья» добавим еще один шаблонный параметр-функтор, предназначенный для создания новых особей, который метод **MakeChild** будет использовать для создания особи. Назовем его CreatureCreator.

Поскольку каждый ребенок является уникальным созданием, в метод MakeChild необходимо передавать экземпляр создателя сущности, предварительно сконфигурированный для создания новой особи с заданными параметрами. Например, для создания детенышей человека Создатель особей должен быть предварительно сконфигурирован именем ребенка, полом и номером свидетельства о рождении.

Для упрощения конструирования создателей сущности стандартный создатель будет просто вызывать конструктор по умолчанию для данного типа особи.

```
// Функтор "Стандартный создатель особи", определяющий стандартный способ
// создания особей
template <typename Creature>
class CStandardCreatureCreator
{
public:
   Creature operator()()const
   {
      return Creature();
   }
};
```

Шаблон класса «Семья» будет подвергнут следующим изменениям:

```
// Шаблон "семья" некоторых существ
template <
   typename Creature,    // Вид особей, составляющих семью
   // Функтор, определяющий пол особи данного вида
   typename GenderDeterminer = CStandardGenderDeterminer<Creature>,
   // Функтор, создающий новую особь данного вида (используется при создании детей)
   typename CreatureCreator = CStandardCreatureCreator<Creature>
>
class CFamily
{
public:
   …
   /* Создание новой особи (ребенка) и добавление его в массив детей семьи
      При попытке вызова данного метода у гомосексуальной пары должно выбрасываться
      исключение std::logic_error
      Создание новой особи происходит при помощи переданного CreatureCreator-а
   */
   Creature MakeChild(CreatureCreator const& creatureCreator = CreatureCreator())
   {
      // Создаем определитель пола
      GenderDeterminer genderDeterminer;
```

```
    // Определяем пол супругов при помощи определителя пола
    Gender gender1 = genderDeterminer(GetSpouse1());
    Gender gender2 = genderDeterminer(GetSpouse2());

    // Запрещаем возможность создания детей в однополой семье
    …
    // Создаем детеныша
    Creature child = creatureCreator();

    // Добавляем детеныша в массив детенышей
    …

    return child;
  }
  …
};
```

Необходимо дописать реализацию данных классов и использовать их в программе, иллюстрирующей создание следующих семей собак, кошек и людей.

Семья собак

Самец – Бобик, самка – Жучка.

Детеныши: Тузик, Шарик, Кнопочка.

Семья кошек

Самец – Васька, самка – Мурка.

Детеныши: Том, Мурзик, Пушок, Машка.

Семья людей

Супруг – Руслан, супруга – Людмила.

Дети: Гвидон, Салтан, Василиса.