Тренин С.А. Информатика. Осень 2014.
Практикум 18. Открытое наследование. Закрытое наследование. Множественное наследование. Полиморфизм классов. Шаблоны алгоритмов.

Вы участвуете в разработке компьютерной игры. В качестве объектов игрового мира выступают средства транспорта различных типов. В следующих четырех пунктах описывается работа с тремя классами, каждый из которых реализует свой тип транспортного средства. Далее дано описание каждого из классов. Требуется сначала реализовать эти классы, затем протестировать их с помощью описанных ниже процедур. Далее требуется выполнить, по крайней мере, одно задание из пяти предлагаемых вариантов и снова протестировать работоспособность программы. Наконец необходимо нарисовать получившуюся в итоге диаграмму классов.

1. Все ТС могут совершать перемещения. Однако каждое перемещение приводит к ухудшению качества ТС. У каждого типа ТС происходят в процессе перемещения свои изменения. Изменения сказываются на дальнейшей работоспособности. Подробнее характеристики каждого типа ТС описаны дальше. Метод совершения перемещения имеет единый формат:
```
virtual bool MakeTrip (double distanceOfTrip);
```
Для базового класса `Vehicle` этот метод является абстрактным.
Сами перемещения можно выполнять непосредственно. Для простоты тестирования далее приведена функция, которая позволяет совершить случайные перемещения с каждым ТС, указатель на которое есть во входном массиве:
```
#include <stdlib.h>

void CommitRandomTrips (MyVector<Vehicle *> &vehicles) {
	for (int i = 0; i < vehicles.Length (); ++i) {
		double randomDistance = double(rand() % 20001) / 10.;
		vehicles[i]->MakeTrip (randomDistance);
	}
}
```
Для корректной работы этой функции требуется в основной программе инициализировать генератор случайных чисел. Например так:
```
int main() {
	srand (0);
```

2. Поскольку в процессе игры пользователь будет получать новые экземпляры ТС, Вам придется реализовать алгоритм ввода информации об этих конкретных экземплярах в массивы ТС пользователя. По одному массиву на каждый описанный тип. Например, если определено три типа ТС: «гужевая повозка», «автомобиль» и «самолет», то соответствующие векторы хранения можно объявить и заполнить так:
```
	//Векторы для хранения экземпляров ТС
	MyVector<Coach> coaches;
	MyVector<Automobile> automobiles;
	MyVector<Aeroplane> aeroplanes;

	//Добавление конкретных объектов
	coaches.PushBack (Coach ("Coach 1", 9.));
	coaches.PushBack (Coach ("Coach 2", 11.));
	coaches.PushBack (Coach ("Coach 3", 10.));
	coaches.PushBack (Coach ("Coach 4", 9.5));
	coaches.PushBack (Coach ("Coach 5"));

	automobiles.PushBack (Automobile ("Automobile 1"));
	automobiles.PushBack (Automobile ("Automobile 2", 90.));
```

```
automobiles.PushBack (Automobile ("Automobile 3", 120.));
automobiles.PushBack (Automobile ("Automobile 4", 150.));

aeroplanes.PushBack (Aeroplane ("Aeroplane 1", 1030.));
aeroplanes.PushBack (Aeroplane ("Aeroplane 2", 560.));
aeroplanes.PushBack (Aeroplane ("Aeroplane 3", 2200.));
```

В конструкторах соответствующих классов первый параметр – имя; второй параметр – начальная скорость. Для параметров по умолчанию объявлены константы:

```
const char DefaultVehicleName[]          = "Untyped vehicle";
const char DefaultCoachName []           = "Default Coach";
const char DefaultAutomobileName []      = "Default Automobile";
const char DefaultAeroplaneName []       = "Default Aeroplane";

const double DefaultVehicleSpeed = -1.;
const double DefaultCoachSpeed = 10.;
const double DefaultAutomobileSpeed = 100.;
const double DefaultAeroplaneSpeed = 500.;
```

3. При выборе ТС пользователь хочет видеть единый список. Вам нужно организовать вывод информации о ТС с использованием массивов указателей на объекты:

```
//Векторы указателей на объекты
MyVector<Vehicle *> coachPointers;
MyVector<Vehicle *> automobilePointers;
MyVector<Vehicle *> aeroplanePointers;

//Инициализация векторов указателей
for (int i = 0; i < coaches.Length (); ++i) {
      coachPointers.PushBack (&coaches[i]);
}

for (int i = 0; i < automobiles.Length (); ++i) {
      automobilePointers.PushBack (&automobiles[i]);
}

for (int i = 0; i < aeroplanes.Length (); ++i) {
      aeroplanePointers.PushBack (&aeroplanes[i]);
}

MyVector<Vehicle *> vehiclePointers = coachPointers + automobilePointers + aeroplanePointers;
```

Тут оператор «+» реализует конкатенацию векторов. Таким образом vehiclePointers – вектор указателей на все транспортные средства.

Вывод нужно организовать в нижеследующем порядке (с помощью, например, функции:

```
void DisplayVehicles (const MyVector<Vehicle *> &vehicles) {
      printf("%s     \t%s\t%s\t%s\n", "Name", "Speed", "Dist", "Time");
      for (int i = 0; i < vehicles.Length (); ++i) {
            printf("%s \t%6.2lf\t%7.2lf\t%6.2lf\n", vehicles[i]->GetName(), vehicles[i]->GetSpeed (),
                                    vehicles[i]->GetTotalDistance (), vehicles[i]->GetTotalTime ());
      }
}
)
```

GetSpeed () – абстрактный метод получения информации о текущей скорости.

3.1 Последовательно из каждого массива, упорядоченные по скорости.

3.2 Единым списком, упорядоченным по текущей скорости.

3.3 Отдельно реализуйте вывод информации о ТС в порядке ввода.

Для упорядочивания массивов указателей можно использовать шаблон алгоритма сортировки выбором:

```
template <class MyType>
void MySwap (MyType &v1, MyType &v2) {
```

```
	MyType v3 = v1;
	v1 = v2;
	v2 = v3;
}

template <class ArrayType, class LessFunctionType>
int FindMinimumIndex (ArrayType &data_array, int beginIndex, int endIndex, LessFunctionType LessFunction) {
	int minimumIndex = beginIndex;
	for (int element_number = beginIndex + 1; element_number <= endIndex; ++element_number) {
		if (LessFunction (data_array[element_number], data_array[minimumIndex])) {
			minimumIndex = element_number;
		}
	}
	return minimumIndex;
}


template <class ArrayType, class LessFunctionType>
void SelectionSort (ArrayType &data_array, int beginIndex, int endIndex, LessFunctionType LessFunction) {
	for (int element_number = beginIndex; element_number < endIndex; ++element_number) {
		int minimumIndex = FindMinimumIndex (data_array, element_number, endIndex, LessFunction);
		MySwap (data_array[minimumIndex], data_array[element_number]);
	}
}
```

С пользовательской функцией сравнения, например:

```
		bool CompareDefault (Vehicle *lhs, Vehicle *rhs) {
			return *lhs < *rhs;
		}
```

или

```
		bool CompareTime (Vehicle *lhs, Vehicle *rhs) {
			return lhs->GetTotalTime () < rhs->GetTotalTime ();
		}
```

4. Опишите несколько классов в соответствие с заданием. Используйте открытое наследование. Нарисуйте на бумаге соответствующую иерархию классов.
   Напоминаю, что открытое наследование семантически соответствует отношению «является» («is-a»).
   Всего требуется описать несколько классов.
   Базовый полиморфный класс «Транспортное средство»;

```
class Vehicle {
public:
	//Конструктор ТС по умолчанию
	Vehicle () : totalDistance (0), totalTime (0), baseSpeed (DefaultVehicleSpeed) {
		SetName (DefaultVehicleName);
	}

	//Параметризованый конструтор ТС
	Vehicle (const char inNameCString[], double inBaseSpeed, int inBasePrice) : totalDistance (0),
totalTime (0), baseSpeed (inBaseSpeed) {
		SetName (inNameCString);
	}

	//Деструктор ТС
	virtual ~Vehicle () {
	}

	//Метод получения информации об имени ТС
	const char * const GetName () const {
		return nameCString;
	}
```

```
	//Метод получения информации о скорости при покупке
	double GetBaseSpeed () const {
		return baseSpeed;
	}

	//Метод получения информации о текущем пробеге
	double GetTotalDistance () const {
		return totalDistance;
	}

	//Метод получения информации о текущем времени эксплуатации
	double GetTotalTime () const {
		return totalTime;
	}

	//Абстрактный метод совершения поездки
	//Возвращает на true, если поезка удачна и false, если случилась авария.
	virtual bool MakeTrip (double distance) = 0;

	//Абстрактный метод получения информации о текущей скорости
	virtual double GetSpeed () const = 0;

	//Оператор сравнения по умолчанию
	bool operator< (Vehicle &rhs) const {
		if (GetSpeed () < rhs.GetSpeed ()) {
			return true;
		}
		return false;
	}

	//Константа - общее максимальное количество символов в имени ТС
	static const int MAX_NAME_LENGTH = 50;

protected:
	//Счетчик пройденного расстояния
	double totalDistance;
	//Счетчик общего времени эксплуатации
	double totalTime;


private:
	//Метод установки имени транспортного средства
	void SetName (const char inNameCString[]) {
		int i = 0;
		for (i = 0; (inNameCString[i] != 0) && (i < MAX_NAME_LENGTH); ++i) {
			nameCString [i] = inNameCString [i];
		}
		nameCString [i] = 0;
	}

	//C - строка для хранения имени
	char nameCString [MAX_NAME_LENGTH];

	//Скорость нового транспортного средства
	double baseSpeed;

};
```

Оператор сравнения может использоваться для реализации способа упорядочивания по умолчанию.

```
	bool operator< (Vehicle &rhs);
```

Все деструкторы полиморфных классов должны быть виртуальными.

```
	virtual ~Vehicle ();
```

Также требуется описать классы нескольких конкретных типов ТС.

Класс «Самолет» (является «Транспортным средством»);

Скорость самолета – не меняется со временем (иначе он не сможет летать).

```
virtual double GetSpeed () const {
	return GetBaseSpeed ();
}
```

Однако авиационное управление запрещает совершать перелет, если с момента последнего технического обслуживания к моменту прибытия в пункт назначения пройдет больше чем MAX_FLY_TIME единиц времени.

```
virtual bool MakeTrip (double distanceOfTrip) {
	double timeOfTrip = distanceOfTrip / GetSpeed () + GetTimeToBoot ();
	if (timeSinceLastRepair + timeOfTrip > MAX_FLY_TIME) {
		return false;
	}
	timeSinceLastRepair += timeOfTrip;
	totalDistance += distanceOfTrip;
	totalTime += timeOfTrip;
	return true;
}
```

timeSinceLastRepair – счетчик времени с момента последнего ТО обновляется специальным методом:

```
void Repair () {
	timeSinceLastRepair = 0;
}
double GetTimeSinceLastRepair () const {
	return timeSinceLastRepair;
}
```

MAX_FLY_TIME – статическое поле класса (может быть изменено по решению авиационного управления).

Кроме того самолет «содержит» бортовой компьютер. Отношение «содержит» в данном примере в иллюстративных целях реализуется через механизм закрытого наследования.

```
const double DefaultTimeToBoot = 0.01;

class Computer {
public:
	Computer () : baseTimeToBoot (DefaultTimeToBoot) {}

	double GetTimeToBoot () {  // возвращает время загрузки компьютера
		return baseTimeToBoot;
	}

protected:
	double baseTimeToBoot;
};
```

В самолете можно выполнить обновление бортового компьютера с помощью метода:

```
void Aeroplane::ComputerUpdate (double newTimeToBoot) {
	baseTimeToBoot = newTimeToBoot;
}
```

Класс «Гужевая повозка» (является «Транспортным средством»);

Пусть скорость повозки падает со временем по закону:

```
	GetBaseSpeed () * exp(-totalTime / 500.);
```

Пусть поездка на повозке считается успешной почти всегда. Однако гужевая повозка не может совершить перемещение на расстояние большее, чем MAX_DISTANCE.

MAX_DISTANCE – статическое поле класса Coach.

Класс «Автомобиль» (является «Транспортным средством»);
Пусть скорость автомобиля падает с пройденным расстоянием по закону:

```
GetBaseSpeed () * exp(-totalDistance / 500.);
```

Реализуйте каждый из трех перечисленных классов. Протестируйте их работоспособность с помощью приведенных функций. Нарисуйте диаграмму классов.
Выполните по крайней мере одно из следующих заданий.

Вариант 1.
По аналогии с шаблоном сортировки выбором реализуйте шаблон сортировки вставками.

Вариант 2.
После выполнения перемещений

```
CommitRandomTrips (vehiclePointers);
```

Найдите самолет с максимальным временем полета момента последнего ТО и выполните для него Repair. Попробуйте использовать для поиска массив указателей на объекты базового типа, для этого потребуется использовать динамическое преобразование, например так:

```
dynamic_cast <Aeroplane *>(aeroplanePointers[0]) -> Repair();
```

Вариант 3.
Модифицируйте код следующим образом.
Пусть от автомобиля как от базового класса наследуют классы «Электромобиль» и «Бензиновый автомобиль».
«Электромобиль» содержит «аккумулятор», который разряжается со временем и может быть перезаряжен до сколько-то процентов исходного заряда. У аккумулятора есть число перезарядок, после которого он отказывается заряжаться и его нужно менять. Если аккумулятор разряжен (или будет разряжен в процессе поездки), то поездка не будет успешной.
«Бензиновый автомобиль» содержит бензобак, который нужно пополнять до поездки на нужное количество литров. Отношения «содержит» здесь и далее моделируйте на Ваше усмотрение с помощью закрытого наследования или композиции. Нарисуйте иерархию наследования.

Вариант 4.
Класс «Автоповозка» (является одновременно «Автомобилем» (если Вы выполнили вариант 3, то «Электромобилем») и «Гужевой повозкой»).
Реализуйте его с использованием механизмов множественного наследования. С помощью виртуального наследования добейтесь, чтобы в состав объектов класса Автоповозка входила только одна копия объекта класса «транспортное средство».
Нарисуйте иерархию наследования.

Вариант 5.
По аналогии со скоростью реализуйте для разных ТС разные алгоритмы вычисления цены.
Нарисуйте иерархию наследования.