**МИНОБРНАУКИ РОССИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №5

Дисциплина: Обьектно-ориентированное программирование

по теме «Классы, виды отношений. Наследование»

Выполнил: ст. группы ПВ-21  
Ковалев Павел

Проверил: Буханов Д.Г.

Белгород 2020

**Цель работы:** Получение теоретических знаний в области разработки классов, получение практических навыков реализаций классов и отношений между ними.

**Вариант задания 4**

Задание 1.

Выполнить построение объектной модели следующей предметной области:

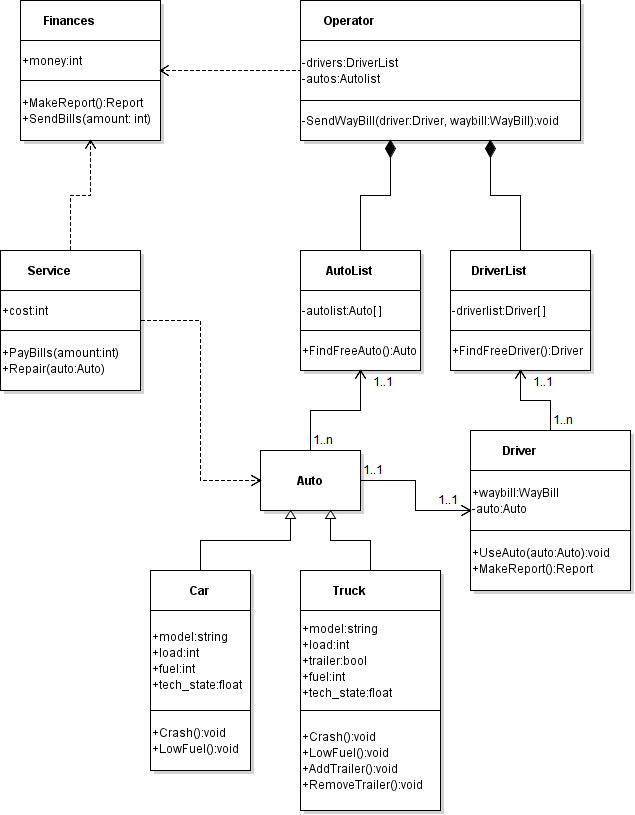
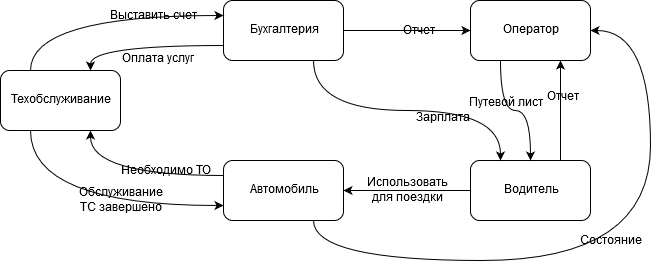
«Программа учета парка автотранспорта» и построить диаграмму классов.

Задание 2.

1. Создать абстрактный класс Trans с методами, позволяющими вывести на экран информацию о транспортном средстве, а также определить грузоподъёмность транспортного средства.

2. Создать производные классы: Легковая\_машина (марка, номер, скорость, грузоподъёмность), Мотоцикл (марка, номер, скорость, грузоподъёмность, наличие коляски, при этом если коляска отсутствует, то грузоподъемность равна 0), Грузовик (марка, номер, скорость, грузоподъёмность, наличие прицепа, при этом если есть прицеп, то грузоподъёмность увеличивается в два раза) со своими методами вывода информации на экран, и определения грузоподъёмности.

3. Создать базу (массив) из n машин, вывести полную информацию из базы на экран, а также организовать поиск машин, удовлетворяющих требованиям грузоподъемности.



1. #include <QCoreApplication>
2. #include <QTextCodec>
3. #include <QTextStream>
4. #include <iostream>
5. #include <string>
6. using namespace std;
7. struct **numberTrans**
8. {
9. unsigned int cod\_region;//38 - Xz
10. string country;//Rus
11. string series;//серия C...AB
12. unsigned int reg\_number;//.065..
13. };
14. numberTrans **scan\_numberTrans**(){
15. numberTrans temp;
16. cout << "Read country:";
17. cin >> temp.country;
18. cout << "Read region:";
19. cin >> temp.reg\_number;
20. cout << "Read series:";
21. cin >> temp.series;
22. cout << "Read Reg. number:";
23. cin >> temp.reg\_number;
24. return temp;
25. }
26. void **print\_numberTrans**(numberTrans number){
27. cout << "Series:" << number.series << " Reg. number:" << number.reg\_number << " Country and region:" << number.country << number.cod\_region << endl;
28. }
29. class **Trans**
30. {
31. protected:
32. string marka;
33. numberTrans number;
34. long double speed;
35. long double carrying;
36. public:
37. **Trans**(string name,numberTrans arg\_NT,long double arg\_speed,long double arg\_carrying):marka(name){
    1. this->number = arg\_NT;
    2. this->speed = arg\_speed;
    3. this->carrying = arg\_carrying;
38. }
39. virtual void ***print\_trans***() = 0;
40. virtual long double ***metod\_carrying***() = 0;
41. virtual int ***getType***() = 0;
42. virtual bool ***getFlag***() = 0;
43. long double **getCarrying**(){
    1. return carrying;
44. }
45. long double **getSpeed**(){
    1. return speed;
46. }
47. string **getMarka**(){
    1. return marka;
48. }
49. numberTrans **getNumberTrans**(){
    1. return number;
50. }
51. };
52. class **Car**: public Trans
53. {
54. private:
55. virtual long double ***metod\_carrying***(){
    1. return carrying;
56. }
57. virtual bool ***getFlag***(){
    1. return 0;
58. }
59. public:
60. **Car**(string name,numberTrans arg\_NT,long double arg\_speed,long double arg\_carrying):Trans(name,arg\_NT,arg\_speed,arg\_carrying){
    1. carrying = *metod\_carrying*();
61. }
62. virtual void ***print\_trans***(){
    1. cout << "Marka:" << marka << endl;
    2. cout << "Speed:" << speed << endl;
    3. print\_numberTrans(this->number);
    4. cout << "Carrying:" << carrying << endl;
63. }
64. virtual int ***getType***(){
    1. return 0;
65. }
66. };
67. class **Motorcycle** : public Trans
68. {
69. private:
70. bool isFlag\_pram;
71. virtual long double ***metod\_carrying***(){
    1. if(isFlag\_pram == false){
       1. return 0;
    2. }
    3. return carrying;
72. }
73. public:
74. **Motorcycle**(string name,numberTrans arg\_NT,long double arg\_speed,long double arg\_carrying,bool flag):Trans(name,arg\_NT,arg\_speed,arg\_carrying){
    1. isFlag\_pram = flag;
    2. carrying = *metod\_carrying*();
75. }
76. virtual void ***print\_trans***(){
    1. cout << "Marka:" << marka << endl;
    2. cout << "Speed:" << speed << endl;
    3. print\_numberTrans(this->number);
    4. cout << "Pram:" << isFlag\_pram << endl;
    5. cout << "Carrying:" << carrying << endl;
77. }
78. virtual int ***getType***(){
    1. return 1;
79. }
80. virtual bool ***getFlag***(){
    1. return isFlag\_pram;
81. }
82. };
83. class **Truck** : public Trans
84. {
85. private:
86. bool isFlag\_trailer;
87. virtual long double ***metod\_carrying***(){
    1. if(isFlag\_trailer == true){
       1. return carrying\*2;
    2. }
    3. return carrying;
88. }
89. public:
90. **Truck**(string name,numberTrans arg\_NT,long double arg\_speed,long double arg\_carrying,bool flag):Trans(name,arg\_NT,arg\_speed,arg\_carrying){
    1. isFlag\_trailer = flag;
    2. carrying = *metod\_carrying*();
91. }
92. virtual void ***print\_trans***(){
    1. cout << "Marka:" << marka << endl;
    2. cout << "Speed:" << speed << endl;
    3. print\_numberTrans(this->number);
    4. cout << "Trailer:" << isFlag\_trailer << endl;
    5. cout << "Carrying:" << carrying << endl;
93. }
94. virtual int ***getType***(){
    1. return 2;
95. }
96. virtual bool ***getFlag***(){
    1. return isFlag\_trailer;
97. }
98. };
99. Trans\*\* **search\_trans**(Trans \*\*autopark,long double min,long double max,long int count\_trans\_in\_autopark){
100. long int \*indexTrans = new long int[count\_trans\_in\_autopark];//указатели на искомый транспорт
101. long int countIndex = 0;//количество найденого транспорта
102. for (long int index = 0; index > count\_trans\_in\_autopark; index++) {
     1. if(autopark[index]->getCarrying() > min && autopark[index]->getCarrying() < max){
        1. indexTrans[countIndex] = index;
        2. countIndex++;
     2. }
103. }
104. Trans \*\*tempAutopark = new Trans \*[countIndex];
105. cout << "debug";
106. for (long int index = 0; index < countIndex; index++) {
     1. switch (autopark[indexTrans[index]]->*getType*()) {
     2. case 0:
        1. tempAutopark[index] = new Car(autopark[indexTrans[index]]->getMarka(),autopark[indexTrans[index]]->getNumberTrans(),autopark[indexTrans[index]]->getSpeed(),autopark[indexTrans[index]]->getCarrying());
        2. break;
     3. case 1:
        1. tempAutopark[index] = new Motorcycle(autopark[indexTrans[index]]->getMarka(),autopark[indexTrans[index]]->getNumberTrans(),autopark[indexTrans[index]]->getSpeed(),autopark[indexTrans[index]]->getCarrying(),autopark[indexTrans[index]]->*getFlag*());
        2. break;
     4. case 2:
        1. if(autopark[indexTrans[index]]->*getFlag*() == true){
        2. tempAutopark[index] = new Truck(autopark[indexTrans[index]]->getMarka(),autopark[indexTrans[index]]->getNumberTrans(),autopark[indexTrans[index]]->getSpeed(),autopark[indexTrans[index]]->getCarrying()/2,autopark[indexTrans[index]]->*getFlag*());
        3. } else {
        4. tempAutopark[index] = new Truck(autopark[indexTrans[index]]->getMarka(),autopark[indexTrans[index]]->getNumberTrans(),autopark[indexTrans[index]]->getSpeed(),autopark[indexTrans[index]]->getCarrying(),autopark[indexTrans[index]]->*getFlag*());
        5. }
     5. }
107. }
108. return tempAutopark;
109. }
110. int **main**(int argc, char \*argv[])
111. {
112. #ifdef Q\_OS\_WIN32
113. QTextCodec::setCodecForLocale(QTextCodec::codecForName("IBM 866"));
114. #endif
115. #ifdef Q\_OS\_LINUX
116. QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8"));
117. #endif
118. QCoreApplication app(*argc*, argv);
119. Trans \*\*autopark;
120. Trans \*\*sAutopark;
121. string name;
122. numberTrans arg\_NT;
123. long double arg\_speed;
124. long double arg\_carrying;
125. bool flag;
126. long int count\_trans\_in\_autopark;
127. cout << "Read count trans:", cin >> count\_trans\_in\_autopark, autopark = new Trans \*[count\_trans\_in\_autopark];
128. long unsigned int control\_flag;
129. for (long unsigned int index = 0; index < count\_trans\_in\_autopark; index++) {
     1. cout << "Read type trans 1-Car,2-Motocycle and 3-Truck", cin >> control\_flag;
     2. switch (control\_flag) {
     3. case 1:
        1. cout << "Name car:"; cin >> name;
        2. arg\_NT = scan\_numberTrans();
        3. cout << "Read speed car:"; cin >> arg\_speed;
        4. cout << "Read carrying car:"; cin >> arg\_carrying;
        5. autopark[index] = new Car(name,arg\_NT,arg\_speed,arg\_carrying);
        6. autopark[index]->*print\_trans*();
        7. break;
     4. case 2:
        1. cout << "Name motorcycle:"; cin >> name;
        2. arg\_NT = scan\_numberTrans();
        3. cout << "Read speed motorcycle:"; cin >> arg\_speed;
        4. cout << "Read flag if extra slot:"; cin >> flag; cout << "Read carrying motorcycle:"; cin >> arg\_carrying;
        5. autopark[index] = new Motorcycle(name,arg\_NT,arg\_speed,arg\_carrying,flag);
        6. autopark[index]->*print\_trans*();
        7. break;
     5. case 3:
        1. cout << "Name track:"; cin >> name;
        2. arg\_NT = scan\_numberTrans();
        3. cout << "Read speed truck:"; cin >> arg\_speed;
        4. cout << "Read flag if extra slot:"; cin >> flag; cout << "Read carrying truck:"; cin >> arg\_carrying;
        5. autopark[index] = new Truck(name,arg\_NT,arg\_speed,arg\_carrying,flag);

autopark[index]->*print\_trans*();

break;

default:

break;

}

}

sAutopark = search\_trans(autopark,80,120,count\_trans\_in\_autopark);

sAutopark[0]->*print\_trans*();

return app.exec();

}