4/5/24, 11:53 PM Person.cpp

sp24-pa6-Andy2Tran/src/Person.cpp

```
1 // Title : Person.h
   // Desc
            : Implementation for Person class
   // Name
              : An Tran
 4
 5
   #include <iostream>
        using std::getline;
 6
 7
        using std::cin;
8
9
   #include <iomanip>
        using std::setw;
10
        using std::left;
11
        using std::right;
12
13
   #include <algorithm>
14
15
   #include "Person.h"
16
17
18 // constructor
   Person::Person() : fName(""), lName(""), address(Address()), pets() {};
19
   Person::Person(const string& fName, const string& lName, const Address& address):
20
    pets() {
21
            setFName(fName);
22
            setLName(lName);
23
            setAddress(address);
24
        };
   Person::Person(const string& fName, const string& lName) : address(Address()), pets()
25
26
            setFName(fName);
27
            setLName(lName);
28
        }:
29
   Person::~Person() {
30
31
        for (Pet* pet : pets) {
32
            delete pet;
33
        };
        pets.clear();
34
   };
35
36
37
   // getters
    string Person::getFName(){
38
39
        transform(fName.begin(), fName.end(), fName.begin(), toupper);
40
        return fName;
   };
41
42
43
   string Person::getLName(){
        transform(lName.begin(), lName.end(), lName.begin(), toupper);
44
45
        return lName:
   };
46
47
   Address& Person::getAddress() {
48
        return address;
49
   }:
50
51
52 std::vector<Pet*>& Person::getPets(){
```

```
53
         return pets;
 54
    };
 55
 56
    // setters
 57
    void Person::setFName(const string& newFName) {
         fName = newFName;
 58
         transform(fName.begin(), fName.end(), fName.begin(), toupper);
 59
    };
 60
 61
    void Person::setLName(const string& newLName) {
 62
         lName = newLName:
 63
 64
         transform(lName.begin(), lName.end(), lName.begin(), toupper);
    };
 65
 66
    void Person::setAddress(const Address& newAddress) {
 67
         address = Address(newAddress.street, newAddress.city, newAddress.state,
 68
    newAddress.zipCode);
 69
    };
 70
    void Person::setStreet(const string& newStreet){
 71
 72
         address.street = newStreet:
         transform(address.street.begin(), address.street.end(), address.street.begin(),
 73
    toupper);
 74
    };
 75
    void Person::setCity(const string& newCity){
 76
 77
         address.city = newCity;
 78
         transform(address.city.begin(), address.city.end(), address.city.begin(),
    toupper);
 79
    };
 80
 81
    void Person::setState(const string& newState){
 82
         address.state = newState;
         transform(address.state.begin(), address.state.end(), address.state.begin(),
 83
    toupper);
 84
    };
 85
 86
    void Person::setZip(const size t& newZipCode){
 87
         address.zipCode = newZipCode;
 88
    };
89
    // other
 90
91
    void operator+(std::vector<Pet*>& pets, Pet* pet) {
 92
         pets.push back(pet);
93
    };
94
95
    void operator-(std::vector<Pet*>& pets, const string& petName) {
96
         auto it = std::find_if(pets.begin(), pets.end(), [&](Pet* pet) {
 97
             return pet->getName() == petName;
98
         }):
         if (it != pets.end()) {
99
100
             delete *it:
101
             pets.erase(it):
102
         } else {
103
             throw std::runtime error("Pet not found.");
104
         };
105
    };
```

4/5/24, 11:53 PM

4/5/24, 11:53 PM Person.cpp

```
106
107
     std::ostream& operator<<(std::ostream& os, Person& person) {
108
         os << left << setw(13) << "FIRST NAME" << ":" << right << setw(21) <<
     person.getFName() << '\n'</pre>
            << left << setw(13) << "LAST NAME" << ":" << right << setw(21) <<</pre>
109
     person.getLName() << '\n'</pre>
            << '\n'
110
            << left << setw(13) << "ADDRESS" << "\n"
111
112
            << person.address
113
            << '\n'
114
            << left << setw(13) << "PETS LIST" << "\n";
115
116
         if (person.pets.empty()) {
117
             os << "NONE\n";
118
         } else {
119
             for (Pet* pet : person.pets) {
                 os << left << setw(13) << "NAME" << ":" << right << setw(21) << pet->
120
     getName() << '\n'</pre>
                     << left << setw(13) << "DOB" << ":" << right << setw(21) << pet->
121
     getDOB().dateString() << '\n'</pre>
122
                     << left << setw(13) << "TYPE" << ":" << right << setw(21) << pet->
     getType() << '\n'</pre>
123
                     << left << setw(13) << "BREED" << ":" << right << setw(21) << pet->
     getBreed() << '\n'</pre>
124
                     << '\n';
125
             };
         }:
126
127
         return os;
128
     };
129
130
     std::istream& operator>>(std::istream& input, Person& person) {
131
         string street;
132
         string city;
133
         string state;
134
         size_t zip;
         getline(input, person.fName);
135
136
         getline(input, person.lName);
137
         getline(input, street);
138
         getline(input, city);
         getline(input, state);
139
140
         input >> zip;
141
         input.ignore();
142
         person.setStreet(street);
143
         person.setCity(city);
144
         person.setState(state);
145
         person.setZip(zip);
146
147
         return input;
148
     };
149
150
     bool Person::searchPet(const string& searchName){
151
         //transform(searchName.begin(), searchName.end(), searchName.begin(), toupper);
152
         for (Pet* pet : pets) {
             if (pet->getName() == searchName) {
153
154
                  return true;
155
             };
156
         };
157
         return false;
```

```
4/5/24, 11:53 PM
 158 };
 159
 160 void Person::addPet() {
 161
          Pet* pet = new Pet();
          std::cin >> *pet;
 162
 163
          if (!searchPet(pet->getName())) {
 164
               this->pets.push back(pet);
 165
          } else {
 166
              delete pet;
 167
              throw std::runtime_error("Pet with same name already exists.");
 168
          };
      };
 169
 170
      void Person::deletePet() {
 171
 172
          string name;
 173
          cin >> name;
          auto it = std::remove_if(pets.begin(), pets.end(), [&](Pet* pet) {
 174
 175
               return pet->getName() == name;
 176
          });
          if (it != pets.end()) {
 177
 178
              delete *it;
              pets.erase(it, pets.end());
 179
 180
          } else {
              throw std::runtime_error("Pet not found.");
 181
 182
          };
 183 };
 184
```