

sp24-pa6-Andy2Tran/src/Person.cpp

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

```
53     return pets;
54 };
55
56 // setters
57 void Person::setFName(const string& newFName) {
58     fName = newFName;
59     transform(fName.begin(), fName.end(), fName.begin(), toupper);
60 };
61
62 void Person::setLName(const string& newLName) {
63     lName = newLName;
64     transform(lName.begin(), lName.end(), lName.begin(), toupper);
65 };
66
67 void Person::setAddress(const Address& newAddress) {
68     address = Address(newAddress.street, newAddress.city, newAddress.state,
69 newAddress.zipCode);
70 };
71
72 void Person::setStreet(const string& newStreet){
73     address.street = newStreet;
74     transform(address.street.begin(), address.street.end(), address.street.begin(),
75 toupper);
76 };
77
78 void Person::setCity(const string& newCity){
79     address.city = newCity;
80     transform(address.city.begin(), address.city.end(), address.city.begin(),
81 toupper);
82 };
83
84 void Person::setState(const string& newState){
85     address.state = newState;
86     transform(address.state.begin(), address.state.end(), address.state.begin(),
87 toupper);
88 };
89
90 void Person::setZip(const size_t& newZipCode){
91     address.zipCode = newZipCode;
92 };
93
94 // other
95 void operator+(std::vector<Pet*>& pets, Pet* pet) {
96     pets.push_back(pet);
97 };
98
99 void operator-(std::vector<Pet*>& pets, const string& petName) {
100     auto it = std::find_if(pets.begin(), pets.end(), [&](Pet* pet) {
101         return pet->getName() == petName;
102     });
103     if (it != pets.end()) {
104         delete *it;
105         pets.erase(it);
106     } else {
107         throw std::runtime_error("Pet not found.");
108     }
109 };
110
111 }
```

```

106
107 std::ostream& operator<<(std::ostream& os, Person& person) {
108     os << left << setw(13) << "FIRST NAME" << ":" << right << setw(21) <<
person.getFName() << '\n'
109     << left << setw(13) << "LAST NAME" << ":" << right << setw(21) <<
person.getLName() << '\n'
110     << '\n'
111     << left << setw(13) << "ADDRESS" << "\n"
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) {
120             os << left << setw(13) << "NAME" << ":" << right << setw(21) << pet->
getName() << '\n'
121             << left << setw(13) << "DOB" << ":" << right << setw(21) << pet->
getDOB().dateString() << '\n'
122             << left << setw(13) << "TYPE" << ":" << right << setw(21) << pet->
getType() << '\n'
123             << left << setw(13) << "BREED" << ":" << right << setw(21) << pet->
getBreed() << '\n'
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;
135     getline(input, person.fName);
136     getline(input, person.lName);
137     getline(input, street);
138     getline(input, city);
139     getline(input, state);
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) {
153         if (pet->getName() == searchName) {
154             return true;
155         };
156     };
157     return false;

```

```
158 };
159
160 void Person::addPet() {
161     Pet* pet = new Pet();
162     std::cin >> *pet;
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
171 void Person::deletePet() {
172     string name;
173     cin >> name;
174     auto it = std::remove_if(pets.begin(), pets.end(), [&](Pet* pet) {
175         return pet->getName() == name;
176     });
177     if (it != pets.end()) {
178         delete *it;
179         pets.erase(it, pets.end());
180     } else {
181         throw std::runtime_error("Pet not found.");
182     };
183 };
184
```