sp24-pa6-Andy2Tran/src/Person.cpp

```cpp
// Title  : Person.h
// Desc   : Implementation for Person class
// Name   : An Tran

#include <iostream>
    using std::getline;
    using std::cin;

#include <iomanip>
    using std::setw;
    using std::left;
    using std::right;

#include <algorithm>

#include "Person.h"

// constructor
Person::Person() : fName(""), lName(""), address(Address()), pets() {};
Person::Person(const string& fName, const string& lName, const Address& address) : pets() {
        setFName(fName);
        setLName(lName);
        setAddress(address);
    };
Person::Person(const string& fName, const string& lName) : address(Address()), pets() {
        setFName(fName);
        setLName(lName);
    };

Person::~Person() {
    for (Pet* pet : pets) {
        delete pet;
    };
    pets.clear();
};

// getters
string Person::getFName(){
    transform(fName.begin(), fName.end(), fName.begin(), toupper);
    return fName;
};

string Person::getLName(){
    transform(lName.begin(), lName.end(), lName.begin(), toupper);
    return lName;
};

Address& Person::getAddress(){
    return address;
};

std::vector<Pet*>& Person::getPets(){
    return pets;
};

// setters
void Person::setFName(const string& newFName) {
    fName = newFName;
    transform(fName.begin(), fName.end(), fName.begin(), toupper);
};

void Person::setLName(const string& newLName) {
    lName = newLName;
    transform(lName.begin(), lName.end(), lName.begin(), toupper);
};

void Person::setAddress(const Address& newAddress) {
    address = Address(newAddress.street, newAddress.city, newAddress.state, newAddress.zipCode);
};

void Person::setStreet(const string& newStreet){
    address.street = newStreet;
    transform(address.street.begin(), address.street.end(), address.street.begin(), toupper);
};

void Person::setCity(const string& newCity){
    address.city = newCity;
    transform(address.city.begin(), address.city.end(), address.city.begin(), toupper);
};

void Person::setState(const string& newState){
    address.state = newState;
    transform(address.state.begin(), address.state.end(), address.state.begin(), toupper);
};

void Person::setZip(const size_t& newZipCode){
    address.zipCode = newZipCode;
};

// other
void operator+(std::vector<Pet*>& pets, Pet* pet) {
    pets.push_back(pet);
};

void operator-(std::vector<Pet*>& pets, const string& petName) {
    auto it = std::find_if(pets.begin(), pets.end(), [&](Pet* pet) {
        return pet->getName() == petName;
    });
    if (it != pets.end()) {
        delete *it;
        pets.erase(it);
    } else {
        throw std::runtime_error("Pet not found.");
    };
};
```

```cpp

std::ostream& operator<<(std::ostream& os, Person& person) {
    os << left << setw(13) << "FIRST NAME" << ":" << right << setw(21) << person.getFName() << '\n'
       << left << setw(13) << "LAST NAME" << ":" << right << setw(21) << person.getLName() << '\n'
       << '\n'
       << left << setw(13) << "ADDRESS" << "\n"
       << person.address
       << '\n'
       << left << setw(13) << "PETS LIST" << "\n";

    if (person.pets.empty()) {
        os << "NONE\n";
    } else {
        for (Pet* pet : person.pets) {
            os << left << setw(13) << "NAME" << ":" << right << setw(21) << pet->getName() << '\n'
               << left << setw(13) << "DOB" << ":" << right << setw(21) << pet->getDOB().dateString() << '\n'
               << left << setw(13) << "TYPE" << ":" << right << setw(21) << pet->getType() << '\n'
               << left << setw(13) << "BREED" << ":" << right << setw(21) << pet->getBreed() << '\n'
               << '\n';
        };
    };
    return os;
};

std::istream& operator>>(std::istream& input, Person& person) {
    string street;
    string city;
    string state;
    size_t zip;
    getline(input, person.fName);
    getline(input, person.lName);
    getline(input, street);
    getline(input, city);
    getline(input, state);
    input >> zip;
    input.ignore();
    person.setStreet(street);
    person.setCity(city);
    person.setState(state);
    person.setZip(zip);

    return input;
};

bool Person::searchPet(const string& searchName){
    //transform(searchName.begin(), searchName.end(), searchName.begin(), toupper);
    for (Pet* pet : pets) {
        if (pet->getName() == searchName) {
            return true;
        };
    };
    return false;
};

void Person::addPet() {
    Pet* pet = new Pet();
    std::cin >> *pet;
    if (!searchPet(pet->getName())) {
        this->pets.push_back(pet);
    } else {
        delete pet;
        throw std::runtime_error("Pet with same name already exists.");
    };
};

void Person::deletePet() {
    string name;
    cin >> name;
    auto it = std::remove_if(pets.begin(), pets.end(), [&](Pet* pet) {
        return pet->getName() == name;
    });
    if (it != pets.end()) {
        delete *it;
        pets.erase(it, pets.end());
    } else {
        throw std::runtime_error("Pet not found.");
    };
};

```