LW: State Park Passport

Overview

Objectives

- Practice with classes, including constructors, getters, and other member functions.
- Working with vectors instead of arrays.
- Managing multiple classes.

Getting Started

- Download the starter code.
  - `StatePark.h` / `StatePark.cpp`: definitions for the `StatePark` class.
  - `Passport.h` / `Passport.cpp`: definitions for the `Passport` class.
  - `Database.h` / `Database.cpp`: definitions for the `Database` class.
  - `Driver.cpp`: a driver file that contains the `main()` function.
  - `park_data.txt` and `camper_data.txt`: text files that store the information that will be loaded into the database.
- You will only need to modify the `.cpp` files.
- You will only need to uncomment commands in `Driver.cpp`.
- Read over the introduction and requirements.
- **You need an 80 on Gradescope** to get completion credit.
- Refer to the vector documentation for tips when using vectors.

Introduction

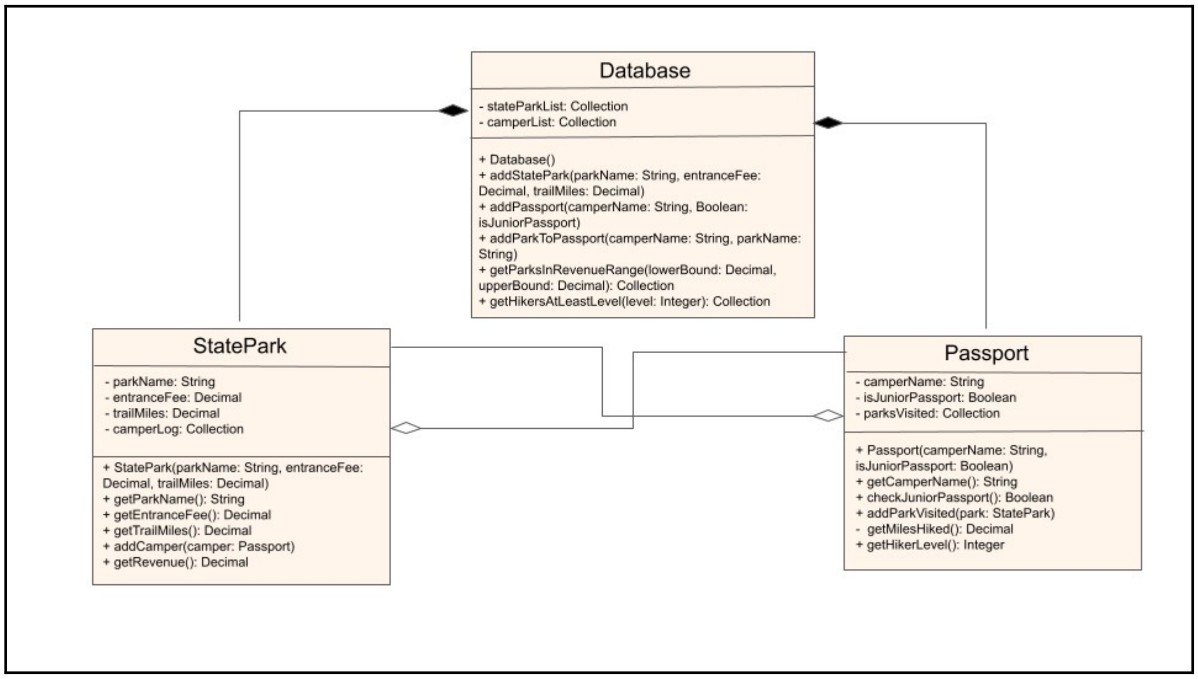
For this labwork, you will implement all the classes necessary for a database so TPWD (Texas Park and Wildlife) can bring their passport program into the 21st century.

You will be implementing a simple "database" modeled by an instance of the `Database` class. This database contains one vector storing pointers to instances of the `StatePark` class and another storing pointers to instances of the `Passport` class. The relationships between `StatePark` and `Passport` objects (e.g., "Alice visited Park X" or "Park Y was visited by Bob") are modeled by pointers between the `StatePark` and `Passport` objects.

Your goal is to correctly implement functions that allow for the database to be constructed. This includes tasks such as adding state parks and passports to the database, creating new relationships between state parks and passports, and other basic functions.

For additional practice, you can also implement some features for the database that allow the user to access certain data, like the revenue of different state parks, and how far different campers have hiked. The `isJuniorPassport` data member is used to denote a Junior passport, whose camper pays 50% of the entrance fee and gets double the miles towards their hiker level. (In real life kids are often free at state parks.)

UML Diagram


Database
- stateParkList: Collection
- camperList: Collection
+ Database()
+ addStatePark(parkName: String, entranceFee: Decimal, trailMiles: Decimal)
+ addPassport(camperName: String, Boolean: isJuniorPassport)
+ addParkToPassport(camperName: String, parkName: String)
+ getParksInRevenueRange(lowerBound: Decimal, upperBound: Decimal): Collection
+ getHikersAtLeastLevel(level: Integer): Collection
StatePark
- parkName: String
- entranceFee: Decimal
- trailMiles: Decimal
- camperLog: Collection
+ StatePark(parkName: String, entranceFee: Decimal, trailMiles: Decimal)
+ getParkName(): String
+ getEntranceFee(): Decimal
+ getTrailMiles(): Decimal
+ addCamper(camper: Passport)
+ getRevenue(): Decimal
Passport
- camperName: String
- isJuniorPassport: Boolean
- parksVisited: Collection
+ Passport(camperName: String, isJuniorPassport: Boolean)
+ getCamperName(): String
+ checkJuniorPassport(): Boolean
+ addParkVisited(park: StatePark)
- getMilesHiked(): Decimal
+ getHikerLevel(): Integer

- The `StatePark` class has four private attributes: `parkName`, `entranceFee`, `trailMiles`, and `camperLog`; and six public methods: `StatePark`, `getParkName`, `getEntranceFee`, `getTrailMiles`, `addCamper`, and `getRevenue`.
- The `Database` class has two private attributes: `stateParkList` and `camperList`; and five public methods: `Database`, `addStatePark`, `addPassport`, `addParkToPassport`, `getParksInRevenueRange`, and `getHikersAtLeastLevel`.
- The `Passport` class has three private attributes: `camperName`, `isJuniorPassport`, and `parksVisited`; five public methods: `Passport`, `getCamperName`, `checkJuniorPassport`, `addParkVisited`, and `getHikerLevel`; and one private method: `getMilesHiked`.

Requirements

Required: Implement the Database

1. Look over the UML diagram in the Background section, as well as the header files `StatePark.h`, `Passport.h`, and `Database.h`, which contain the class definitions and the function declarations you must implement.
2. Implement the constructors for the `StatePark`, `Passport`, and `Database` classes in `StatePark.cpp`, `Passport.cpp`, and `Database.cpp` respectively. **Make sure to use a member initializer list and initialize any vectors to `{}`**.
3. In `StatePark.cpp`:
    1. Implement the getter functions `getParkName()`, `getEntranceFee()`, and `getTrailMiles()`.
    2. Implement the function `addCamper()`, which takes in a pointer to a `Passport` object as a parameter and adds it to the end of the vector `camperLog`.
4. In `Passport.cpp`:
    1. Implement the getter functions `getCamperName()` and `checkJuniorPassport()`.
    2. Implement the function `addParkVisited()`, which takes in a pointer to a `StatePark` object as a parameter and adds it to the end of the vector `parksVisited`. Additionally, it should use the `addCamper()` function to add (a pointer to) itself to the `camperLog` of that state park. (Hint: How do you get a pointer to the current object?)
5. In `Database.cpp`:
    1. You **do not** have to implement the destructor.
    2. Implement the function `addStatePark()`, which should dynamically create a new `StatePark` object and add a pointer to the object to the member vector `stateParkList`.
    3. Implement the function `addPassport()`, which should dynamically create a new `Passport` object and add a pointer to the object to the member vector `camperList`.
    4. Implement the function `addParkToPassport()`, which takes in two parameters `camperName` and `parkName`. The function should search the member vectors of the database for a match for both, then use the `addParkVisited()` function to add the park to the camper's list of visited parks. If one and or both of the parameters are not found, do not make any changes to the database.
6. **In `Driver.cpp`, uncomment out the corresponding sections of code**.

Optional: Implement Park Revenue

1. In `StatePark.cpp`:
    1. Implement the function `getRevenue()`, which should go through the vector camperLog and calculate how much revenue the park made. Remember that the entrance fee for a Junior Passport holder is half of the full fee.
2. In `Database.cpp`:
    1. Implement the function `getParksInRevenueRange()`, which should return a string vector of the names of all the parks whose revenue falls between the two parameters `lowerBound` and `upperBound` (including the boundary values). If no parks are in the range, return an empty vector.
3. In `Driver.cpp`, uncomment out the corresponding sections of code.

Optional: Implement Hiker Levels

1. In `Passport.cpp`:
    1. Implement the (private) function `getMilesHiked()`, which should go through the vector `parksVisited` and calculate how many miles the camper has hiked.
    2. Implement the function `getHikerLevel()`, which should calculate what level hiker the camper is (each `100` miles equals one level, e.g., a camper who has hiked 451 miles is a level 4 hiker). Remember that each mile hiked by a Junior Passport holder counts double when calculating hiker level.
2. In `Database.cpp`:
    1. Implement the function `getHikersAtLeastLevel()`, which should return a string vector of the names of all the campers who have reached at least the level represented by the int parameter passed in.
3. In `Driver.cpp`, uncomment out the corresponding sections of code.