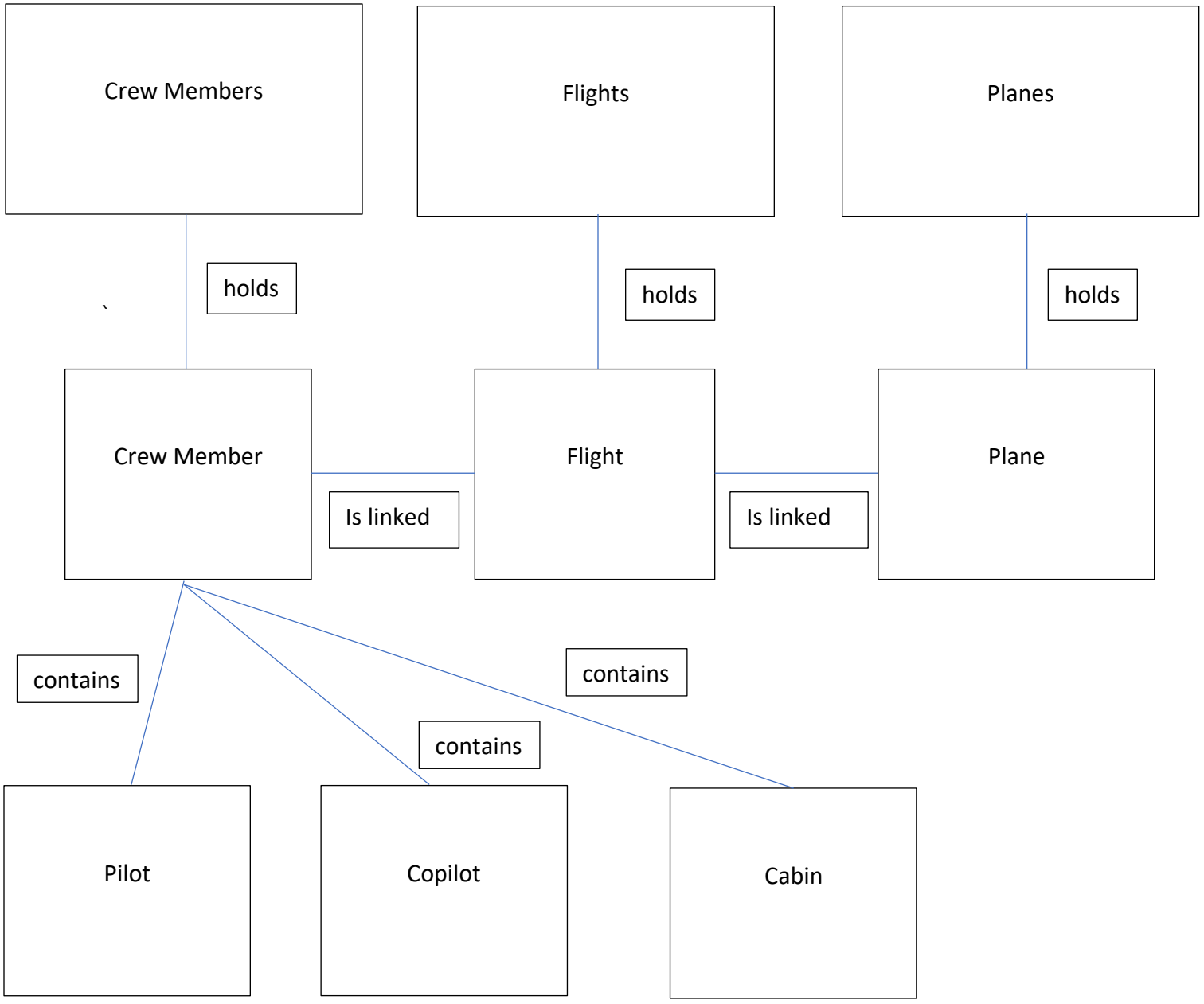
Mean Green Airline Database

(Revised Design 2.0)

Rohan Maheshwari

CSCE 1040.002 – HW5

Class Design



Crew Member

Attributes:
- String name
- Int memberID
- String memberType

Methods:
SetName()
GetName()
Set memberID()
GetmemberID()
Set memberType()
Get memberType()
EditMember()
Virtual printinfo()

Pilot

Attributes:
- String Ratingcode
- Int flighthours

Methods:
Set/get ratingcode
Set/get flighthours
Void editPilot()
Printinfo()

Copilot

Attributes:
- String Ratingcode
- Int flighthours

Methods:
Set/get ratingcode
Set/get flighthours
Void editCopilot()
Printinfo()

Cabin

Attributes:
- String position

Methods:
Set/get position
editCabin()
printinfo()

Flight

Attributes:
- Int tailnumber
- Int numpilots
- Int numcabin
- Vector <int> pilots
- Vector <int> cabinCrew
- String flightstart
- String flightend
- string startairportcode
- string endairportcode
- int numpassenger
- string flightstatus
- Boolean crewRestriction
- int flightid;
- double flightlength;
- int starttimenumber;
- int endtimenumber;
-

Methods:

```
void setTailNumber(string t_number);
      string getTailNumber();
      void setNumPilots(int n_pilots);
      int getNumPilots();
      void setNumCabins(int n_cabins);
      int getNumCabins();
      void setStartairportcode(string s_code);
      string getStartairportcode();
      void setEndairportcode(string e_code);
      string getEndairportcode();
      void setNumpassengers(int n_passengers);
      int getNumpassengers();
      void setFlightstatus(string f_status);
      string getFlightstatus();
      void setCrewrestriction(bool c_restriction);
      bool getCrewrestriction();
      void setFlightID(int f_id);
      int getFlightID();
      void setPilots(vector <int> Pilots);
      vector <int> getPilots();
      void setCabinCrew(vector <int> cabinCrew);
      vector <int> getCabinCrew();
      void setStarttime(string start_time);
      string getStarttime();
      void setEndtime(string end_time);
      string getEndtime();
      void setflightlength(double f_length);
      double getflightlength();
        void setstarttimenumber(int s_time);
      int getstarttimenumber();
      void setendtimenumber(int e_time);
      int getendtimenumber();
```

Plane

Attributes:
- String make
- Int modelnumber
- Int tailnumber
- Int numseats
- Int range
- Int minCCrew

Methods:
Setmake()
Getmake()
SetModelNumber()
GetModelNumber()
SetTailNumber()
GetTailNumber()
SetNumSeats()
GetNumSeats()
SetRange()
GetRange()
SetMinCCrew()
GetMinCrew()

Crew Members

Attributes:
Vector <crewmember*> members
-membercount
vector <int> pilotID;
vector <int> copilotID;
vector <int> cabinID;

Methods:

SetName()
GetName()
Set memberID()
GetmemberID()
Set memberType()
Get memberType()
AddMember()
DeleteMember()
FindMemberToEdit()
PrintMembers()
PrintAMember()
void setpilotID(vector <int> pID)
vector <int> getpilotID()
void setcopilotID(vector <int> coID)
vector <int> getcopilotID()
void setcabinID(vector <int> cID);
vector <int> getcabinID()

Flights

Attributes:
- Vector <flight> flight_list

Methods:

AddFlight()
DeleteFlight()
FindFlightToEdit()
PrintFlights()
PrintAFlight()
void editFlight();
int flightCount();
void updateflightstatus();
void loaddata();
void savedata();
void printAPlaneSchedule();
void printAMemberSchedule();
void deleteExpiredFlights();
void setflightlist(vector <flight> flist);
vector <flight> getflightlist();

Planes

Attributes:
Vector<Plane> plane_list
Vector <string> tailnumber
Int planeCount
Constructor: set plane count = 0

Methods:
AddPlane()
DeletePlane()
FindPlaneToEdit()
PrintPlane()
PrintAPlane()
Void setplanes(vector <plane> plist);
vector <plane> getplanes();
void settailnumber(vector <string> tnumber);
vector <string> gettailnumber();

Algorithm

Plane

EditPlane()

1. Prompt user for the property they would like to edit
2. According to response, prompt the user to enter a new value for the desired property
3. Update the variable for that property with the new input

Planes

AddPlane()

1. Prompt user for each type of data
2. Push back vector size by the current value of plane count
3. Using the mutator functions for each variable, set all variables for the index of the plane count
4. Increment plane count by 1 for the next plane that will be entered

FindingPlaneToEdit() <- for the other functions in plane *take in ID to perform function*

1. Prompt the user for the tail number
2. Perform a loop search to compare the inputted tail number with the tail number of each plane
3. If match is found, break the loop while retaining the amount of iterations done before breaking
4. Call the editPlane() function on the plane at the retained index for user to edit plane properties

DeletePlane()

1. Prompt the user for the tail number
2. Perform a loop to compare the inputted tail number with the tail number of each plane
3. If match is found, break the loop while retaining the amount of iterations done before breaking
4. Use the vector erase function to erase the element at the retained index
5. All following elements will be shifted down automatically to account for this change

SearchPlane() *take in criteria and return planeID*
bool matchFound = false

1. Prompt user for each attribute EXCEPT tail number

2. Using a loop, compare the first inputted attribute with the same attribute in all other planes
3. If the attribute is matched, enter a nested if statement that checks the next inputted attribute with the attribute of the same plane index
4. Continue entering nested if statements as each attribute is matched
5. If all attributes are met, break the loop while retaining the index of the plane before breaking and set the boolean matchFound to true
6. If boolean is true, print out the tail number of the plane at the retained index
7. If an attribute is not matched at a nested if statement, the looping will continue until all planes have been checked
8. If boolean remains false, print a statement informing the user that no such plane exists

PrintPlanes()

1. Use a loop to print out all tail numbers of each planeusing the accessor function in the vector plane list

PrintAPlane()

1. Prompt user to enter a tail number
2. Using a loop, compare inputted tail number with the tail number of each plane in the system
3. If match is found, break the loop while retaining the index of the plane before breaking
4. Using the accessor functions, print out all attributes of the plane at the retained index

Crew Member

EditMember()

1. Prompt user for the property they would like to edit
2. Based on response, prompt the user to enter a new value for that property
3. Update the property with the new input

Crew Members

AddMember()

1. Prompt user to input each attribute
2. Use the vector push back function with the current value of the memberCount
3. Using the mutator functions, set all variables for the index of the member count
4. Increment memberCount by 1 for the next added member

DeleteMember()

1. Prompt user for memberID
2. Using a loop, compare inputted member ID with the member ID of each member
3. If match found, break out of loop while retaining the index value before breaking
4. Using the vector erase function, erase the element at the retained index
5. All elements above the deleted elements will shift down automatically

FindMemberToEdit()

1. Prompt user to enter member ID
2. Using a loop, compare each member ID with the member ID of all other members
3. If match is found, break out of loop while retaining the index value before breaking
4. Call the editMember() on the member with the retained index to edit member properties

SearchMember()
boolean matchFound = false

1. Prompt the user enter all attributes EXCEPT member ID
2. Using a loop, compare the first inputted attribute with the same attribute in all other members
3. If attribute is matched, enter a nested if statement that checks the next inputted attribute with the attribute of the same plane index
4. Continue entering nested if statements as each attribute is matched
5. If all attributes are matched, break the loop while retaining the index of the member before breaking and set the matchFound boolean to true.
6. If boolean is true, print out the memberID of the member at the retained index
7. If an attribute is not mtached at a nested if statement, the looping will continue until all members have been checked
8. If the boolean remains false, print a statement informing the user that no such member exists

PrintMembers()

1. Use a loop to print out all member IDs using the accessor function in vector member_list

PrintAMember()

1. Prompt user to enter a member ID
2. Using a loop, compare inputted member ID with the member ID of each member in the system
3. If match is found, break the loop while retaining the index of the member before breaking

4. Using the accessor functions, print out all attributes of the member at the retained index

Flight

EditFlight()

1. Prompt the user for the property they would like to edit
2. Based on response, prompt the user to enter a new value for that property
3. Allow for range checking to ensure valid response
4. Update the property with the new input

Flights

AddFlight()

1. Prompt user to enter all attributes
2. Start with starting date/time and ending date/time as some attributes require range checking with respect to flight length, set crewRestrictions to true if flight length is past 8 hours
3. Depending on the value of boolean crewRestrictions (by using accessor function), range check number of pilots and cabin crew using an if and while statement accordingly to comply with regulations
4. Range check to make sure planes and crew members exist when assigning them to the flight
5. Use the vector push back function with the current value of the flightCount
6. Use the mutator functions to set all variables to the index of flightCount
7. Increment flightCount by 1 for the next inputted flight

DeleteFlight()

1. Prompt user to input a flight ID
2. Using a loop, compare inputted flight ID with the flight ID of all other flights in the system
3. If match is found, break out of loop while retaining value of index before breaking
4. Using the vector erase function, erase the element at the retained index value
5. All elements following the deleted element will automatically shift down one index

FindFlightToEdit()

1. Prompt user to enter flight ID
2. Using a loop, compare each flight ID with the flight ID of all other flights

3. If match is found, break out of loop while retaining the value of the index before breaking
4. Call the editFlight() function on the flight with the retained index to edit flight properties
5. Prompt user to enter member ID
6. Using a loop, compare each member ID with the member ID of all other members
7. If match is found, break out of loop while retaining the index value before breaking
8. Call the editMember() on the member with the retained index to edit member properties

PrintFlights()

1. Using a loop, print out the flight IDs of all fights using the accessor function in vector flight_list

PrintAFlight()

1. Prompt the user to enter a flight ID
2. Using a loop, compare the inputted flight ID with the flight ID of each flight in the system
3. If a match is found, break the loop while retaining the index of the flight before breaking
4. Using the accessor functions, print out all attributes of the member at the retained index
    a. For date/time, print out each variable independently and format as how a date and time would appear

updateflightstatus()

1- Using a loop, go through all the flights
2- Check if the current time is past the flight end time
3- If so, set the flight status to “completed”

loaddata()

1- Declare a ifstream variable
2- Using a loop, read in all data for each attribute (crew member, plane, flight) for each element in the vector
    a. For crewmembers, read in a label to identify which type of member it is before reading in data
3- Set all data to respective mutator functions

savedata()

1- Declare an ofstream variable
2- Using a loop, write out all data for each attribute (crew member, plane, flight) for each element in the vector
    a. For Crew members, write out a lavel to indicate crew member type for organized saving

PrintAPlaneSchedule()

1 - Ask for plane ID
2 – Use a loop to search all flights that have the plane ID that was inputted
3 – If found, print all attributes of that flight

PrintACrewMemberSchedule()

1 – Ask for member ID
2 – Use a loop to search all flights that have the member ID that was inputted
3 – If found, print all attributes of that flight

DeleteExpiredFlights()

1 – Use a loop to search for all flights that have a flight status of “completed” or “cancelled”
2 – If found, use the vector erase function to remove that flight entry

Design Report

With the class inheritance implementation, I realized that there are more affected functions of the database than I predicted. For example, I realized that edit functions are dependent upon the new class inheritance as the properties to edit are different for some of the derived classes. Additionally, the use of the virtual functions pushed me to use new object identification methods such as typeid. Interestingly, my load and save functions were also affected by the inheritance implementation. For these functions, I had to add labels to differentiate between derived class types in order to read in the proper information for the right type of variable. The HW5 experience has taught me a lot about double checking all aspects of the program to catch any affected areas that I might have not thought to be affected. Although these affected areas might not cause compilation errors, they may produce runtime and logic errors that could be much more difficult to find later. Overall, my understanding of the objective of the assignment was slowly improving and I feel more confident about class inheritance, iterators, extern methods, and virtual functions after this assignment.