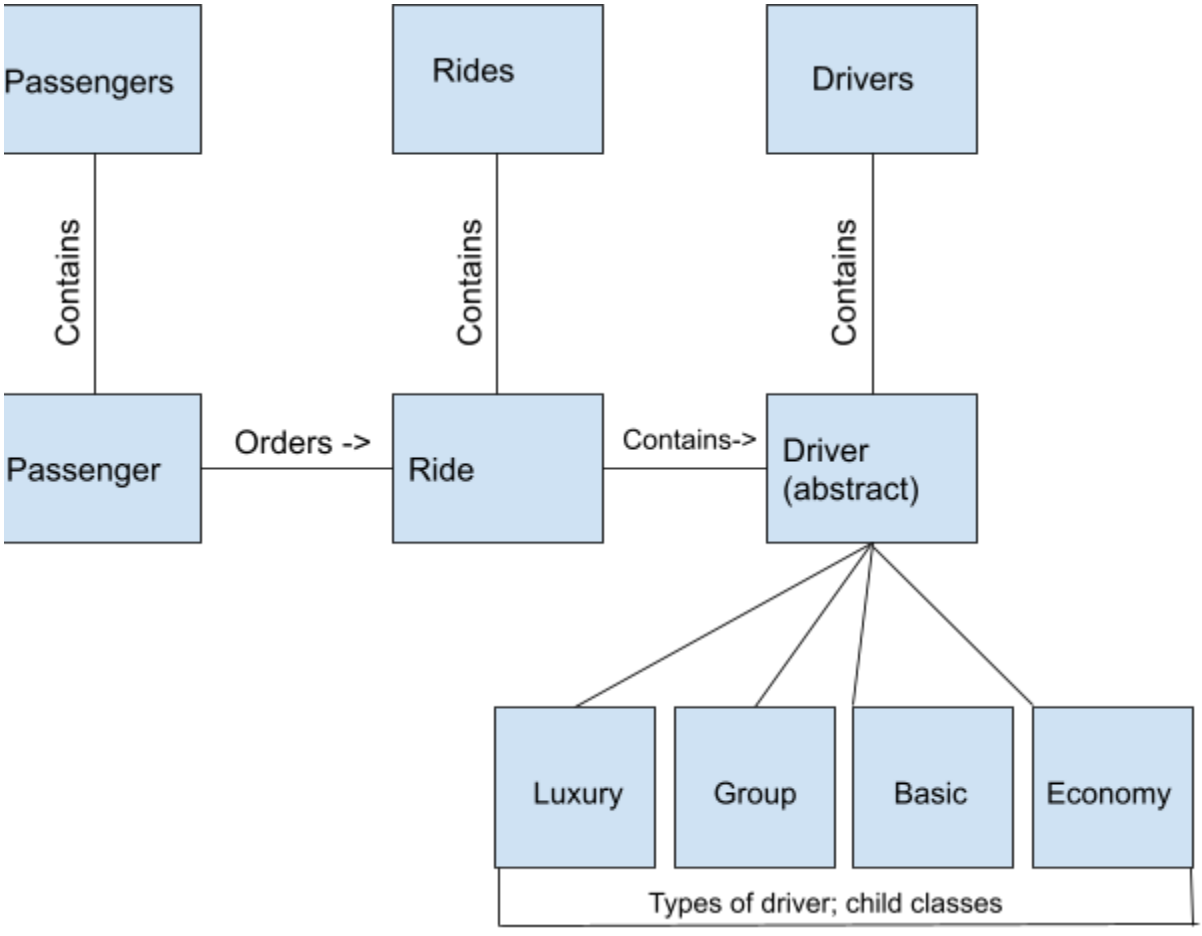
CSCE 1040.002

Homework 4, by Krish Kohir

Mean Green EagleLyft Design Document

Class Relationships



Class Contents:

Passenger:

- Name (string)

- ID (int)
- Payment preference (string)
- Handicapped (bool)
- Required rating (float)
- Has Pets (bool)
- Size of party
-
- Get/Set methods for all variables
- Print

Ride:

- ID (int)
- Pickup Location (string)
- Pickup Time (Time value)
- Dropoff Location (string)
- Drop off time (Time value)
- Size of party (int)
- Has Pets (bool)
- Needs handicap (bool)
- Status (string)
- Rating by customer (float)
- Passenger (*passenger)
- Driver (*driver)
- Passenger ID
- Driver ID
- Get/Set methods for all variables
- Set passenger
- Set driver

Driver:

- Name (string)
- ID (int)
- Capacity (int)
- Handicap Capable (bool)
- Vehicle Type (string)
- Available (bool)
- Driver Rating (float)
- Pets Allowed (bool)
- Notes (string)

- Type of driver (int)
- Get/Set methods for all variables
- Get/Set luggage space (pure virtual)
- Print function (virtual)
- Empty destructor (virtual)

Economy (derived from Driver):

- Luggage Space (int)
- Price Per Mile = 1.00 (const double)
- Max capacity = 2 (const int)
- Min capacity = 1 (const int)
- Get/Set methods for all variables
- Print function (overridden)
- Destructor (overridden)

Basic (derived from Driver):

- Luggage Space (int)
- Price Per Mile = 1.50 (const double)
- Max capacity = 4 (const int)
- Min capacity = 2 (const int)
- Get/Set methods for all variables
- Print function (overridden)
- Destructor (overridden)

Group (derived from Driver):

- Luggage Space (int)
- Price Per Mile = 2.20 (const double)
- Max capacity = 7 (const int)
- Min capacity = 5 (const int)
- Get/Set methods for all variables
- Print function (overridden)
- Destructor (overridden)

Luxury (derived from Driver):

- Luggage Space (int)
- Amenities (vector of strings)
- Price Per Mile = 1.00 (const double)
- Max capacity = 2 (const int)
- Min capacity = 1 (const int)
- Get/Set methods for all variables

- Add Amenity
- Add Amenities
- GetAmenity (string)
- Print function (overridden)
- Destructor (overridden)

Passengers:

- Vector of passengers (passengers)
- Add passenger
- Edit passenger (parameter int ID)
- Delete passenger
- Find passenger
- Get passenger
- Print passenger
- Print all passengers
- Find and Print a passenger
- Store data
- Load data

Rides:

- Vector of rides (rides)
- Add ride
- Edit ride
- Delete ride
- Compare ride times
- Update ride
- Update driver availability
- Find ride
- Print all rides
- Print a ride
- Activate rides
- Print all Pending rides
- Print all Active Rides
- Print all Completed Rides
- Print all Canceled Rides
- Print all Rides for passenger
- Print passenger schedule
- Print all Rides for driver
- Print driver schedule

- Delete all canceled rides
- Delete all completed rides
- Store data
- Load data
-

Drivers:

- Vector of drivers (drivers)
- Get drivers
- Add driver
- Edit driver (parameter int ID)
- Delete driver
- Find driver
- Get driver
- Print all drivers
- Print a driver
- Store data
- Load data

Function pseudo code

Passenger:

- **Print passenger**
    - Print out all variables.

Ride:

- **Set passenger (parameter passenger)**
    - Set the passenger, and change ride variables to reflect passenger variables
- **Set driver(parameter driver)**
    - Check to see if driver meets requirements.
    - If so, allocates memory based on driver
    - Sets driver.
- **Print ()**
    - Prints out variables

Driver:

- **Print driver**
    - Print out all variables.

Passengers:

- **Add passenger**
    - Create passenger object
    - Prompt for each variable
    - Populate passenger variables
    - Add passenger to collection
    - Store data
- **Edit passenger**
    - Prompt for each passenger variable
    - Prompt for passenger ID
    - Find passenger through ID (return index), and use set methods to edit data
    - Store data
- **Delete passenger (parameter ID)**
    - Find passenger through ID (return index), remove passenger from collection and delete object.
    - Store data
- **Find passenger (parameter ID)**
    - Loop through collection, if ID of passenger matches argument ID, return index of passenger in vector.
    - If nothing returned, print not found, return null.
- **Get passenger (parameter ID)**
    - Use find passenger to return index of passenger.
    - Return a reference of the passenger
    - If nothing returned, print not found, return null.
- **Print All Passengers**
    - Loop through collection, use Print on all passenger objects
- **Find and Print passenger**
    - Find passenger through ID (return passenger), and use Print on object
- **Store data**
    - Loop through collection, write each variable into passenger text file
- **Load data**

    - Create while loop to see if data can be taken. If so, create a passenger object and assign variables with data from text file. Add passenger to collection

Drivers:

- **Get drivers**
    - Return the drivers vector
- **Add driver**
    - Create driver object
    - Prompt for each variable
    - Populate driver variables
    - Checks if driver is
    - Add driver to collection
    - Store data
- **Edit driver**
    - Prompt for each driver variable
    - Prompt for driver ID
    - Find driver through ID (return driver), and use set methods to edit data
    - Checks if new capacity is supported by type of driver
    - Store data
- **Delete driver**
    - Remove driver from collection and delete object
    - Store data
- **Find driver (parameter ID)**
    - Loop through collection, if ID of driver matches argument ID, return driver object
    - If nothing returned, print not found, return null.
- **Get Driver (parameter ID)**
    - Use find driver to return index of driver.
    - Return a reference of the driver.
    - If nothing returned, print not found, return null.
- **Print driver**
    - Print out all variables with brief captions, utilize end lines.
- **Print All Drivers**
    - Loop through collection, use Print on all driver objects
- **Find and Print driver**
    - Find driver through ID (return driver), and use Print on object.
- **Store data**

    - Loop through collection, write each variable into driver text file
- **Load data**
    - Create while loop to see if data can be taken. If so, create a driver object and assign variables with data from text file. Checks to see if driver is luxury to add amenities. Add driver to collection

Rides:

- **Add ride (parameters Passengers, Drivers)**
    - Create ride object
    - Prompt for each variable that isn't dependent on passenger.
    - Populate the ride variables.
    - Ask for passenger ID. Find passenger in passengers collection using find and add to ride vector. Edit ride variables dependent on passenger variables.
    - Loop through ride collection for all rides that have same driver object.
    - If time variables of other rides contradict with current ride object, or if driver doesn't meet pet or handicap standards, or if driver doesn't have enough seats, or if driver doesn't meet rating, print fail and delete ride object.
    - Else, populate driver variable.
    - Add ride to collection
    - Store data
- **Edit ride**
    - Prompt for each ride variable
    - Prompt for ride ID
    - Find ride through ID (return ride), and use set methods to edit data
    - Store data
- **Delete ride**
    - Remove ride from collection and delete object
    - Store data
- **Compare ride times (parameter Ride ride1, Ride ride2)**
    - Takes two rides and compares which one starts first
    - Used for sort() function
- **Update rides**
    - Change ride status to canceled or complete
    - Sees if current status is compatible. If so, adds dropOffTime and changes status.
    - Store data

- **Update driver availability (parameter drivers class)**
    - Ask for driver ID to get driver
    - Loop through each ride. If ride is active and has same driver, set driver availability to false.
    - Else update driver availability to true
    - Store driver data
- **Activate rides**
    - Loop through collection, if current time is past ride's pick up time, set status to active
    - Store data
- **Find ride (parameter ID)**
    - Loop through collection, if ID of ride matches ride ID, return ride object
    - If nothing returned, print not found, return null.
- **Print All rides**
    - Loop through collection, use Print on all ride objects
- **Print ride**
    - Find ride through ID (return ride), and use Print on object.
- **Find ride (parameter ID)**
    - Loop through collection, if current time is past start time
    - If nothing returned, print not found, return null.
- **Print all Pending Rides**
    - Loop through collection, use Print on rides with pending status
- **Print all Active Rides**
    - Loop through collection, use Print on rides with active status
- **Print all Completed Rides**
    - Loop through collection, use Print on rides with completed status
- **Print all Canceled Rides**
    - Loop through collection, use Print on rides with canceled status
- **Print passenger schedule**
    - Ask for passenger ID to get passenger
    - Loop through collection, try to see if ride has passenger.
    - If so, add to vector.
    - Sort vector using sort function()
    - Use print on each element in vector
- **Print all Rides for passenger**
    - Loop through collection, try to find passenger in each ride object. If found, use Print on ride.
- **Print driver schedule**
    - Ask for driver ID to get passenger

    - Loop through collection, try to see if ride has driver.
    - If so, add to vector.
    - Sort vector using sort function()
    - Use print on each element in vector
- **Print all Rides for driver**
    - Loop through collection, try to find driver in each ride object. If found, use Print on ride.
- **Delete canceled rides**
    - Loop through collection, if ride is canceled than use delete.
    - Store data
- **Delete completed rides**
    - Loop through collection, if ride is completed than use delete.
    - Store data
- **Store data**
    - Loop through collection, write each variable into ride text file excluding pointers.
- **Load data (parameter passengers and drivers)**
    - Create while loop to see if data can be taken. If so, create a ride object and assign variables with data from text file. Use ID's for passenger and driver to add pointers to ride. Add ride to collection.

Economy:

- **Print driver**
    - Print out all variables. Overrides the parent print and adds driver type, luggage space, and price per mile

Basic:

- **Print driver**
    - Print out all variables. Overrides the parent print and adds driver type, luggage space, and price per mile

Group:

- **Print driver**
    - Print out all variables. Overrides the parent print and adds driver type, luggage space, and price per mile

Luxury:

- **Print driver**
    - Print out all variables. Overrides the parent print and adds driver type, luggage space, price per mile, and all amenities
- **Add amenities**

- Repeatedly prompts user for amenities until they enter stop.
- Adds amenities to driver

Report:

Overall, the implementation of child classes into homework 4 was not much of an issue. The individual child classes don't bring up enough change to drastically impact my code. However, that isn't to say that nothing changed in the other classes. For example, I turned the original driver class into an abstract class, as I knew that only the child classes should be instantiated. However, my rides class had an allocated pointer to a regular driver object, which became an issue. Small hiccups like these were common, but not frustrating.

But, there was one part of the code I found confusing. I declared all the drivers as driver pointers, even if they were initialized as child classes. This meant that I couldn't access their constants, as the code saw the pointers as regular drivers. I tried dereferencing the pointer, but it still didn't fix the issue. I realized that I would have to typecast the object, but dynamic casting only works from child to parent class. Finally, I learned that I had to static cast in order to access the child classes variables. I later realized that I could have made virtual functions, but this solution also works.