Module 1 Assessment Exercise

The assessment you are about to complete will aid you in validating your understanding of the module 1 objectives:

- Variables and data types
- Conditional and iteration logic
- Object-Oriented Programming
- File I/O and Collections
- Unit Testing
- Debugging
- Version control

It will also give you practice with coding assessments you may encounter during the job interview process.

Overview

You may use any resource available in the classroom except another Human Being. This is an individual effort. It is of no value in assessing your skills if you get assistance in completing it from your fellow students.

Remember: The objective of these exercises is for **YOU** to find out how much **YOU** learned in module 1 and help us determine where we need focus on improving **YOUR** weaknesses.

If you get assistance from anyone to complete any of the exercises, you have rendered the objective pointless.

If you any questions, see your instructor.

Any unit tests written should also be error free and run successfully.

Instructions

1. Pull from your class repository like you do for code every day. The assessment will be in the `module-1` folder of your class repro folder.
2. Open the project in the `Module_1_Coding_Assessment` folder.
3. You should see the typical folder structure for a project with a skeleton application program (*main()* or *Main()*) method.
4. Any files you create should be placed in the correct folders within the project.
5. Be sure to push your final effort when you are done.
6. Schedule time with your instructor to review your effort.

Choose any one of the following problems for your assessment.

Teller Machine

1. Create a new class that represents a *Teller Machine*.
2. Add a *manufacturer, deposits, withdrawals, and balance* attribute/property to the Teller Machine class:
    - `manufacturer`: indicates the manufacturer name for the teller machine.
    - `deposits`: indicates the total amount that has been deposited into the machine.
    - `withdrawals`: indicates the total amount that has been withdrawn from the machine.
    - `balance`: indicates the net difference between `deposits` and `withdrawals`.
3. Create a constructor that accepts `manufacturer`, `deposits`, and `withdrawals`.
4. Instantiate an object (or objects) in *main()* or *Main()* and use the object(s) to test your methods.
5. Create a method that checks to see if a string, `cardNumber`, is a valid card. The method only returns `true` under the following conditions:
    - if the `cardNumber` begins with a `5` and has 16 digits
    - if the `cardNumber` begins with a `4` and has 13 or 16 digits
    - if the `cardNumber` begins with a `3` and is followed by a `4` or a `7`.
6. Override the *ToString*()*/toString()* method and have it return `"ATM - {manufacturer} - {balance}"` where `{manufacturer}` and `{balance}` are placeholders for the actual values. i.e. the values from the object should be shown in the string where the `{variable}` is indicated.
7. Implement unit tests to validate the functionality of:
    - the balance calculation
    - the valid card number method
8. In the main program class, within the main method, read in the provided csv file `TellerInput.csv` and use it to populate a list of *Teller Machine* objects.
9. Add up the total balance for all of the teller machines in the list and print it to the screen.

Hotel Reservation

1. Create a new class that represents a *Hotel Reservation*.
2. Add a *name, number of nights, and estimated total* attribute/property to the Hotel Reservation class:
    - `name`: indicates the name on the reservation
    - `number of nights`: indicates how many nights the reservation is for
    - `estimated total`: indicates the estimated total using `number of nights` times a daily rate of $59.99
3. Create a constructor that accepts `name` and `number of nights`.
4. Instantiate an object (or objects) in *main()* or *Main()* and use the object(s) to test your methods.
5. Create a method that calculates the actual total using two *bool/boolean* inputs: `requiresCleaning` and `usedMinibar`.
    - if the minibar was used, a fee of $12.99 is added to the estimated total
    - if the room requires cleaning, a fee of $34.99 is added to the estimated total
    - the cleaning fee is doubled if the minibar was used
6. Override the *ToString()/toString()* method and have it return `"RESERVATION - {name} - {estimated total}"` where `{name}` and `{estimated total}` are placeholders for the actual values. i.e. the values from the object should be shown in the string where the `{variable}` is indicated.
7. Implement unit tests to validate the functionality of:
    - the estimated total calculation
    - the actual total method
8. In the main program class, within the main method, read in the provided csv file `HotelInput.csv` and use it to populate a list of *Hotel Reservation* objects.
9. Add up the estimated total for all of the hotel reservations in the list and print it to the screen.

Flower Shop Order

1. Create a new class that represents a *Flower Shop Order*.
2. Add a *bouquet type, number of roses, and subtotal* attribute/property to the Flower Shop Order class:
    - `bouquet type`: indicates the type of bouquet
    - `number of roses`: indicates the number of roses added to the bouquet
    - `subtotal`: indicates the order subtotal before shipping by adding $19.99 for the standard bouquet plus $2.99 for each rose
3. Create a constructor that accepts `bouquet type` and `number of roses`.
4. Instantiate an object (or objects) in *main()* or *Main()* and use the object(s) to test your methods.
5. Create a method that calculates the delivery total using a *bool/boolean* input `sameDayDelivery` and a string `zipCode`.
    - the delivery fee is $3.99 if the zipcode falls between 20000 and 29999 (+$5.99 for same day delivery)
    - the delivery fee is $6.99 if the zipcode falls between 30000 and 39999 (+$5.99 for same day delivery)
    - the delivery fee is waived ($0.00) if the zipcode falls between 10000 and 19999
    - all other zipcodes cost $19.99 (same day delivery is not an option)
6. Override the *ToString()/toString()* method and have it return `"ORDER - {bouquet type} - {number of roses} roses - {subtotal}"` where `{bouquet type}`, `{number of roses}`, and `{subtotal}` are placeholders for the actual values. i.e. the values from the object should be shown in the string where the `{variable}` is indicated.
7. Implement unit tests to validate the functionality of:
    - the correct subtotal calculation
    - the delivery fee calculation
8. In the main program class, within the main method, read in the provided csv file `FlowerInput.csv` and use it to populate a list of *Flower Shop Order* objects.
9. Add up the subtotal total for all of the orders in the list and print it to the screen.

Car

1. Create a new class that represents a *Car*.
2. Add a *year, make, age, and is classic car* attribute/property to the Car class:
    - `year`: indicates the year that the car was manufactured
    - `make`: indicates the make of the car
    - `is classic car`: indicates if the car is a classic car
    - `age`: indicates the age (in years) of the car from the current year
3. Create a constructor that accepts `year`, `make`, and `is classic car`.
4. Instantiate an object (or objects) in *main()* or *Main()* and use the object(s) to test your methods.
5. Create a method that determines if the car must receive an e-check using an input `int yearToCheck`.
    - Return `true` under the following conditions:
        - even-model year vehicles must be tested if `yearToCheck` is even
        - odd-model year vehicles must be tested if `yearToCheck` is odd
    - Return `false` if an e-check is not needed or the car is exempt
        - a vehicle is exempt if it is under 4 years of age
        - a vehicle is exempt if it is over 25 years of age
        - classic cars are always exempt
6. Override the *ToString()/toString()* method and have it return `"CAR - {year} - {make}"` where `{year}`, `{make}` are placeholders for the actual values. i.e. the values from the object should be shown in the string where the `{variable}` is indicated.
7. Implement unit tests to validate the functionality of:
    - the age calculation
    - the e-check method
8. In the main program class, within the main method, read in the provided csv file `CarInput.csv` and use it to populate a list of *Car* objects.
9. Add up the age for all of the cars in the list and print it to the screen.

Movie Rental

1. Create a new class that represents a *Movie Rental*.
2. Add a *title, format, is premium movie, and rental price* attribute/property to the Movie Rental class:
    - `title`: indicates the title of the movie
    - `format`: indicates the format of the movie (e.g. VHS, DVD, BluRay)
    - `is premium movie`: indicates if the movie is a premium movie (these cost more)
    - `rental price`: indicates the rental price (VHS $0.99, DVD $1.99, BluRay $2.99). Premium movies add an additional $1.00 to the rental price
3. Create a constructor that accepts `title`, `format`, and `is premium movie`.
4. Instantiate an object (or objects) in *main()* or *Main()* and use the object(s) to test your methods.
5. Create a method that determines the movie's late fee using an input `int daysLate`.
    - Return $0.00 if `daysLate` is equal to 0.
    - Return $1.99 if `daysLate` is equal to 1.
    - Return $3.99 if `daysLate` is equal to 2.
    - Return $19.99 if `daysLate` 2 or more.
6. Override the *ToString()/toString()* method and have it return `"MOVIE - {title} - {format} {rental price}"` where `{title}`, `{format}`, and `{rental price}` are placeholders for the actual values. i.e. the values from the object should be shown in the string where the `{variable}` is indicated.
7. Implement unit tests to validate the functionality of:
    - the rental price calculation
    - the late fee calculation
8. In the main program class, within the main method, read in the provided csv file `MovieInput.csv` and use it to populate a list of *Movie Rental* objects.
9. Add up the rental price for all of the movies in the list and print it to the screen.

Concert Ticket

1. Create a new class that represents a *Concert Ticket*.
2. Add a *ticket type, t-shirt purchased, is VIP, and ticket price* attribute/property to the Concert Ticket class:
    - `ticket type`: indicates the class of ticket (Orchestra, Balcony, General, Promo)
    - `t-shirt purchased`: indicates if t-shirt was purchased with the ticket ("Yes" or "No"); Add $20 to the ticket price is a t-shirt was purchased.
    - `is VIP`: indicates ticket is for a VIP (true-VIP, false=not a VIP)
    - `ticket price`: indicates the price of the ticket based on type (Orchestra $100.00, Balcony $79.99, General $50.00, Promo $0). VIPs get 40% off the regular price of their ticket
3. Create a constructor that accepts `ticket type`, `t-shirt purchaned`, and `is VIP`.
4. Instantiate an object (or objects) in *main()* or *Main()* and use the object(s) to test your methods.
5. Create a method that determines an adjusted ticket price based on how many days before the event a ticket was purchased. The method will receive `int daysBefore`.
    - Return the ticket price if `daysBefore` is equal to 0.
    - Return ticket price times .95 if `daysBefore` is between 1 and 10.
    - Return ticket price times .90 if `daysBefore` is between 11 and 20.
    - Return ticket price times .80 if `daysBefore` is between 20 and 30.
    - Return ticket price times .75 if `daysBefore` is greater than 30.
6. Override the *ToString()/toString()* method and have it return `"TICKET - {ticket type} - {ticket price}"` where `{ticket type}` and `{ticket price}`are placeholders for the actual values. i.e. the values from the object should be shown in the string where the `{variable}` is indicated.
7. Implement unit tests to validate the functionality of:
    - the ticket price calculation
    - the adjusted ticket price calculation
8. In the main program class, within the main method, read in the provided csv file `TicketInput.csv` and use it to populate a list of *Concert Ticket* objects.
9. Add up the rental price for all of the movies in the list and print it to the screen.