
Week #5

Objective: For students to get some practice of:

- To understand the concept of Constructors and Destructors
  . (No argument and with Argument Constructors)
- Problem solving with classes
  - Writing formal Class description
    - .1 Determining data types for member data
    - .2 Member Function prototypes
  - Translating class descriptions into class declarations
    - .1 Use of private, public keywords.
    - .2 Data/information hiding
  - Working with objects in various applications
    - .1 Calling member functions
    - .2 Rum-time/actual arguments
    - .3 Default-arguments

Theory:

A class can be viewed as a customized 'struct' that **encapsulates data** and **function**. Format of a class definition:

```
class your_class_name
{           member_access_specifier:
                data members;
            member_access_specifier:
                member_functions();
};
```

When cin and cout are used to perform I/O, actually objects are created from istream and ostream respectively that has been defined in iostream header file.

Building a Class

```
#include <iostream>
using namespace std;
class smallobj                                  //declare a class
{
private:
    int somedata;                               //class data
public:
    void setdata(int d)                         //member function to set data
        { somedata = d; }
    void showdata()                 //member function to display data
        {
        cout << "Data is " << somedata << endl; }
};

void main()
{
smallobj s1, s2;                    //define two objects of class smallobj
s1.setdata(1066);                   //call member function to set data
s2.setdata(1776);
```

```
s1.showdata();                          //call member function to display data
s2.showdata();
}
```

Lab Exercise:

Q #1.

Design then implement a class to represent a **Flight**. A Flight has a *flight number*, a *source*, a *destination* and a *number of available seats*. The class should have:

a. A **constructor** to initialize the 4 instance variables. You have to shorten the name of the source and the destination to 3 characters only if it is longer than 3 characters by a call to the method in the 'j' part.

b. An **overloaded constructor** to initialize the *flight number* and the *number of available seats* instance variables only.
(**NOTE:** Initialize the *source* and the *destination* instance variables to empty string, i.e." ")

c. An **overloaded constructor** to initialize the *flight number* instance variable only.
(**NOTE:** Initialize the *source* and the *destination* instance variables to empty string; and the *number of available seats* to zero)

d. One **accessor** method for each one of the 4 instance variables.

e. One **mutator** method for each one of the 4 instance variables **except** the *flight number* instance variable.

f. A **method** `public void reserve(int numberOfSeats)` to reserve seats on the flight.
(**NOTE:** You have to check that there is enough number of seats to reserve)

g. A **method** `public void cancel(int numberOfSeats)` to cancel one or more reservations

h. A `toString` method to easily return the flight information as follows:

```
Flight No: 1234
From: KAR
To: LAH
```

i. An `equals` method to compare 2 flights.
(**NOTE:** 2 Flights considered being equal if they have the same flight number)

j. The following method:

```
private String shortAndCapital (String name) {
  if (name.length() <= 3) {
    return name.toUpperCase();
  } else {
    return name.substring(0,3).toUpperCase();
  }
}
```

Create the object of the flight class and apply the defined methods.

Output:

```
Enter No of seats:
4
Enter Source:
kharachi
Enter Destination:
lahore
Reserve Seats : 4
Avalaible Seats Seats : 336
Enter Data Of Flight 2:
Enter Flight NO:
124
Enter No of seats:
5
Enter Source:
lahore
Enter Destination:
karachi
Reserve Seats : 5
Avalaible Seats Seats : 335
Flight 1 Info
Flight No:123
Source:KHA
Destination:LAH
Flight 2 Info
Flight No:124
Source:LAH
Destination:KAR
Two Flights are not Same
```

Q#2.

Implement a class *Car,* that has the following characteristics:

a) *brandName*,
b) *priceNew,* which represents the price of the car when it was new,
c) *color*, and
d) *odometer*, which is milo meter shows number of mileage travelled by car

The class should have:

A. A method *getPriceAfterUse()* which should return the price of the car after being used according to the following formula:

$$\text{car price after being used} = priceNew \times (1 - \frac{odemeter}{600{,}000})$$

B. A method *updateMilage(double traveledDistance)* that changes the current state of the car by increasing its mileage, and
C. A method *outputDetails()* that will output to the screen all the information of the car, i.e., brand name, price new, price used, color, and odometer.

Write a test class for the *Car* class above. You are required to do the followings:

a. Create an object of type *Car*.
b. Assign any valid values to the instance variables of the object created in '**A**'.
c. Use the method *getPriceAfterUse* on the object created in '**A**' then output the result to the screen.
d. Use the method *updateMilage* on the object created in '**A**' by passing a valid value.
e. Do part '**C**' again.
f. Use the method *outputDetails* on the object created in '**A**'.

Output:

```
Enter Brand:
toyota
Enter Color:
blue
Enter New Price:
13444
Enter Od0meter:
54
Brand:toyota
Color:blue
New Price:13444.0
Odometer:54.0
Price After use:13442.79004
Brand:toyota
Color:blue
New Price:13444.0
Odometer:10054.0
Price After use:13218.723373333334
Brand:toyota
Color:blue
New Price:13444.0
Odometer:30054.0
Price After use:12770.590040000001
BUILD SUCCESSFUL (total time: 27 seconds)
```

Q#3.

Coffee Outlet runs a catalog business. It sells only one type of coffee beans. The company sells the coffee in 2-lb bags only and the price of a single 2-lb bag is $5.50. when a customer places an order, the company ships the order in boxes. The boxes come in 3 sizes with 3 different costs:

	Large box	Medium box	Small box
Capacity	20 bags	10 bags	5 bags
Cost	$1.80	$1.00	$0.60

The order is shipped using the least number boxes. For example, the order of 52 bags will be shipped in 2 boxes: 2 large boxes, 1medium and 1 small.

Develop an application that computes the total cost of an order.

Sample out put:

```
Number of Bags Ordered: 52
The Cost of Order: $ 286.00

Boxes Used:
        2 Large  - $3.60
        1 Medium - $1.00
        1 Small  - $0.60

Your total cost is: $ 291.20
```