Laboratory 7 - Classes

Your name:	Zach Evans
Your email address:	evansz2@students.wwu.edu
Your student ID number:	W01006113

Lab Objectives

To gain experience with

- Discovering classes
- Implementing your own classes
- Separating class interface and implementation
- Encapsulation
- Object construction
- Designing and implementing accessor and mutator member functions
- Object-oriented design

R1. Discovering classes

In each of the following examples, locate and describe those conceptually related procedures and data elements that can be combined into a single class.

```
/* NAME    : best_profit.cpp
   PURPOSE : Calculates the best city for making a profit as a housing
                       contractor.
*/

#include <iostream>
#include <string>
using namespace std;

int main()
{
   bool mdata = true;
   string response = "";
   double avg_sale_price = 0.0;
   double avg_building_cost = 0.0;
   double avg_profit = 0.0;
   double best_profit = 0.0;
   string city = "";
   string best_city = "";

   while (mdata)
   {  cout << "Average building cost: ";
      cin >> avg_building_cost;

      cout << "Average sale price: ";
      cin >> avg_sale_price;

      cout << "City : ";
      cin >> city;
          avg_profit = avg_sale_price - avg_building_cost;

      if ( avg_profit> best_profit)
      {   best_profit = avg_profit;
          best_city = city;
      }

      cout << "More data (y/n)? ";
      cin >> response;
      if (response != "y")
      {   mdata = false;
      }
   }

   cout << "The best city to be a builder in is: "<<city << "\n";
   return 0;
}
```

```
Make "City" a class.

class City
{
public:
   City();
   City(string name, double buildingCost, double salePrice);

   string getName() const;
   double getAvgCost() const;
   double getAvgSalePrice() const;
   double avgProfit();

private:

   string cityName;
   double avgBuildingCost;
   double avgSalePrice;

};
```

```
/*  NAME:    perk_tst.cpp
    PURPOSE: Main program to record information about a properties perk test
*/

#include <iostream>
#include <string>
#include "rand.h"
using namespace std;

const int lower_limit = 7;
const int upper_limit = 120;
const int lower_acceptable = 10;
const int upper_acceptable = 105;
```

```
void perc_dimension(int& width, int& length)
{
        width = rand_int(1,20);
        length = rand_int(1,20);
}

void air_temp(int& temperature)
{
        temperature = rand_int(0,120);
}

void perc_rate(int& prate)
{
        prate = rand_int(lower_limit,upper_limit);
}

void print_report(string site,string city, int length, int width,int prate,int temperature)
{
        cout << "Site        " << " " << site << "\n" <<
            "City        " << " " << city << "\n" <<
            "Dimensions  " << " " << length <<" X "<<width<< "\n" <<
            "Air Temp    " << " " << temperature<< "\n" <<
                        "Perc Rate   " << " " << prate <<"\n";
        if (prate >= lower_acceptable && prate <= upper_acceptable)
                cout <<"Perc test is ok!\n";
        else
                cout <<"Perc test is not ok!\n";

}


int main()
{
   string site = "";
   string city = "";
   int temperature = 0;
   int width = 0;
   int length = 0;
   int percrate = 0;

   cout << "Site ? ";
   cin >> site;
   cout << "City ? ";
   cin >> city;
   rand_seed();
   perc_dimension(width,length);
   air_temp(temperature);
   perc_rate(percrate);
   print_report(site,city,length,width,percrate,temperature);
   return 0;
}
```

```
"Site" should be a class

class Site
{
public:

   Site();
   Site(string siteName, string siteCity);
   Site(string siteName, string siteCity, int siteTemp, int siteWidth, int siteLength sitePercRate);


   string getName() const;
   string getCity() const;
   int getTemp() const;
   int getWidth() const;
   int getLength() const;
   int getRate() const;

   void print();

private:
   string name;
   string city;
   int temperature;
   int width;
   int length;
   int percRate;

};
```

R2. Interfaces

An object is constructed and it's data accessed and modified only through the use of a class' member functions. These functions define a 'black box' interface to the member data. Provide a constructor and the accessors and mutators necessary to perform the identified services from within a main function using the classes you defined in the of three preceeding examples.

best_profit

```
(I think my answers for the previous part are what you were looking for here, but I
already wrote these implementations so I'm just going to keep them here.)

City::City()
{
   cityName = "";
   avgBuildingCost = 0.0;
   avgSalePrice = 0.0;
}

City::City(string name, double buildingCost, double salePrice)
{
  cityName = name;
  avgBuildingCost = buildingCost;
  avgSalePrice = salePrice;
}

string City::getName() const
{
  return cityName;
}

double City::getAvgCost() const
{
  return avgBuildingCost;
}

double City::getAvgSalePrice() const
{
  return avgSalePrice;
}

double City::avgProfit()
{
 return avgSalePrice - avgBuildingCost;
}
```

Perc Test

```
Site::Site()
{
   name = "";
   city = "";
   temperature = 0;
   width = 0;
   length = 0;
   percRate = 0;

}

Site::Site(string siteName, string siteCity)
{
  name = siteName;
  city = siteCity;
  temperature = 0;
  width = 0;
  length = 0;
  percRate = 0;

}

Site::Site(string siteName, string siteCity, int siteTemp, int siteWidth, int siteLength, int sitePercRate)
{
  name = siteName;
  city = siteCity;
  temperature = siteTemp;
  width = sideWidth;
  length = siteLength;
  percRate = sitePercRate;
}


Site::string getName() const { return name; }
Site::string getCity() const { return city; }
Site::int getTemp() const { return temperature; )
Site::int getWidth() const { return width; }
Site::int getLength() const { return length; }
Site::int getRate() const {return percRate; }

void Site::print()
{

    cout << "Site        " << " " << name << "\n" <<
            "City        " << " " << city << "\n" <<
            "Dimensions  " << " " << length <<" X "<<width<< "\n" <<
            "Air Temp    " << " " << temperature<< "\n" <<
            "Perc Rate   " << " " << percRate <<"\n";


}
```

R3. Encapsulation

Consider the following class declaration.

```
class Customer
{
public:
   Customer(string name, string address, string city, string state, string zipcode);

   void increase_limit(double amount);
   string get_name()  const;
   string get_address() const;
   string get_city()  const;
   string get_state() const;
   double credit_limit;
private:
   string name;
   string address;
   string city;
   string state;
   string zipcode;
}
```

`class Customer` fails to effectively encapsulate a data member, thereby risking a runtime error due to corrupted data. Locate the problem(s) and propose a solution.

```
put "double credit_limit" in the "private" section and add setters and getters
for it in the public section.
```

One of the data members is properly encapsulated but its value cannot be detected. What is it, and what member function might be added later that might use it?

```
"zipcode" cannot be detected. Add "string get_zipcode() const;" to the public
section.
```

P1. Member functions

Suppose that the Customer class is going to be expanded to keep track of total year-to-date purchases.Declare data fields, accessors and mutators which will keep track of the total number of sales and total sales for each instance of Customer.

```
class Customer
{
public:
   Customer(string name, string address, string city, string state, string zipcode);

   void increase_limit(double amount);
   string get_name()  const;
   string get_address() const;
   string get_city()  const;
   string get_state() const;
   double credit_limit;
   /*your work goes here

   */
private:
   string name;
   string address;
   string city;
   string state;
   string zipcode;
   /* and here

   */
  ;
}
```

Write a program that tests the features of your Customer class.

```
#include <iostream>

using namespace std;

class Customer
{
public:
    Customer(string customer_name, string customer_address, string customer_city, string customer_state, string customer_zipcode);

    string get_name()  const;
    string get_address() const;
    string get_city()  const;
    string get_state() const;
    string get_zipcode() const;
    int getNumPurchases() const;
    double getTotalSales() const;
    double getCreditLimit() const;

    void addPurchase(double purchasePrice);
    void setCreditLimit(double num);
    void increase_limit(double amount);

private:
    string name;
    string address;
    string city;
    string state;
    string zipcode;
    int numPurchases;
    double totalSales;
    double creditLimit;
};

Customer::Customer(string customer_name, string customer_address, string customer_city, string customer_state, string customer_zipcode){
    name = customer_name;
    address = customer_address;
    city = customer_city;
    state = customer_state;
    zipcode = customer_zipcode;
    numPurchases = 0;
    totalSales = 0.0;
    creditLimit = 0.0;

}

string Customer::get_name()  const { return name; }
string Customer::get_address() const { return address; }
string Customer::get_city()  const { return city; }
string Customer::get_state() const { return state; }
string Customer::get_zipcode() const { return zipcode; }
int Customer::getNumPurchases() const { return numPurchases; }
double Customer::getTotalSales() const { return totalSales; }
double Customer::getCreditLimit() const { return creditLimit; }


void Customer::addPurchase(double purchasePrice){
    numPurchases++;
    totalSales += purchasePrice;
}

void Customer::setCreditLimit(double num){
    creditLimit = num;

}
void Customer::increase_limit(double amount){
    creditLimit += amount;
}


int main(int argc, const char * argv[])
{

    Customer newCustomer = Customer("Bob", "1234 Main St.", "Anytown", "WA", "12345");

    newCustomer.setCreditLimit(1000);

    cout << newCustomer.get_name() << " has made " << newCustomer.getNumPurchases() << " purchases for a total of $" << newCustomer.getTotalSales() << ".\n";

    newCustomer.addPurchase(12.50);
    newCustomer.addPurchase(100.25);
    newCustomer.addPurchase(12.00);

    cout << newCustomer.get_name() << " has made " << newCustomer.getNumPurchases() << " purchases for a total of $" << newCustomer.getTotalSales() << ".\n";

    return 0;
}
```

Suppose that management now wants to automate the customer management system to detect when a new purchase cannot be made because the credit limit will be exceeded. You'll need to declare or modify data members, constructors, accessors and mutators to expand the `Customer` class to return a value specifying that a new purchase would exceed the credit limit. This can be calculated by passing the purchase price into a member function, adding this to the unpaid balance and comparing it to the credit line.

Enhance your `Customer` class to fulfill the new requirements. Write a program that tests the new feature. Insert print statements into your code to provide a trace of your program's execution of a series of sales and a subsequent reorder.

```
#include <iostream>

using namespace std;

class Customer
{
public:
    Customer(string customer_name, string customer_address, string customer_city, string customer_state, string customer_zipcode);

    string get_name()  const;
    string get_address() const;
    string get_city()  const;
    string get_state() const;
    string get_zipcode() const;
    int getNumPurchases() const;
    double getTotalSales() const;
    double getCreditLimit() const;
    double getUnpaidBalance() const;

    void addPurchase(double purchasePrice);
    void setCreditLimit(double num);
    void increase_limit(double amount);

private:
    string name;
    string address;
    string city;
    string state;
    string zipcode;
    int numPurchases;
    double totalSales;
    double creditLimit;
    double unpaidBalance;
};

Customer::Customer(string customer_name, string customer_address, string customer_city, string customer_state, string customer_zipcode){
    name = customer_name;
    address = customer_address;
    city = customer_city;
    state = customer_state;
    zipcode = customer_zipcode;
    numPurchases = 0;
    totalSales = 0.0;
    creditLimit = 0.0;

}

string Customer::get_name()  const { return name; }
string Customer::get_address() const { return address; }
string Customer::get_city()  const { return city; }
string Customer::get_state() const { return state; }
string Customer::get_zipcode() const { return zipcode; }
int Customer::getNumPurchases() const { return numPurchases; }
double Customer::getTotalSales() const { return totalSales; }
double Customer::getCreditLimit() const { return creditLimit; }
double Customer::getUnpaidBalance() const { return unpaidBalance; }


void Customer::addPurchase(double purchasePrice){


    if ((unpaidBalance + purchasePrice) <= creditLimit){
        numPurchases++;
        totalSales += purchasePrice;
        unpaidBalance += purchasePrice;
    }
    else{
        cout << "Credit limit exceeded. This purchase cannot be made.\n";
    }
}

void Customer::setCreditLimit(double num){
    creditLimit = num;

}
void Customer::increase_limit(double amount){
    creditLimit += amount;
}

void printCustomerBalance(Customer &inCustomer){
    cout << inCustomer.get_name() << " has an unpaid balance of $" << inCustomer.getUnpaidBalance()
         << " and a credit limit of $" << inCustomer.getCreditLimit() << ",\n";
}


int main(int argc, const char * argv[])
{

    Customer newCustomer = Customer("Bob", "1234 Main St.", "Anytown", "WA", "12345");

    newCustomer.setCreditLimit(1000);

    newCustomer.addPurchase(500.0);
    printCustomerBalance(newCustomer);

    newCustomer.addPurchase(450.0);
    printCustomerBalance(newCustomer);

    newCustomer.addPurchase(100.0); //Should not go through
    printCustomerBalance(newCustomer);
```

Don't forget to print to a pdf file and upload the file when you're finished.