Classes/Encapsulation (Practice Code)

Program 1 (code example using VM):

public/private access (need object), function return value

Sven wants to create a program that allows him to enter the color of a fish and decide if he should buy it from the pet store. Assume the pet store has one fish of a specific color (green) and size (medium, indicated by the number 2) left.

```
computer$ g++ rev_one.cpp
computer$ ./a.out
Enter fish color:
blue
-Don't get the fish!

computer$ ./a.out
Enter fish color:
green
-Get the fish!
```

```
#include <iostream>
#include <string>

//NOT using: using namespace std;

class Fish{

  std::string color;
  int size; //1=small, 2=medium, 3=large

public:
  Fish(std::string color, int size)
  {
    this->color=color;
    this->size=size;
  }

  std::string get_color()
  {
    return color;
  }

  int get_size()
  {
    return size;
  }
```

```
};

class Person{

public:

  //Person decides whether to pick fish or not based on color. The fish object (with all its characteristics) is passed into the person function.
  bool pick_fish(Fish f, std::string choice) //Returns boolean. true get, false dont get
  {
      bool ret=false;

      if(f.get_color()==choice) //if the given fish matches the color choice given by the person, get the fish
      {
        ret=true; //we indicate this by changing the value of the boolean to true
      }

      return ret;
  }

};

int main(int argc, char **argv)
{
  //using the constructor to initialize our Fish object in the pet store, hardcoding green for this example (since the pet store has a green fish)
  Fish f1("green", 2);
  Person p1;

  std::string answer;
  std::cout<< "Enter fish color: " <<std::endl; //Color of fish the person wants
  std::cin >> answer;


  //notice even though the function pick_fish is public, we still need
  //a Person object to use it (same for attributes). Same for get_size().  Same thing like Java
//remember pick_fish returns a boolean
  if(p1.pick_fish(f1, answer)&& f1.get_size()==2) //matching color and medium size (since that's what the pet store has)
  {
    std::cout <<"-Get the fish!"<<std::endl;
  }

  else
  {
    std::cout << "-Don't get the fish!" <<std::endl;
  }
}
```

Program 2:

Objects interacting, calling a function on an object returned by another function (hint: this is helpful for HW 2). REMEMBER, we are using the objects we create to call functions in the class the object was created from.

Create a program that keeps track of the goals soccer players make during a game.

```
computer$ g++ rev_two.cpp
computer$ ./a.out
Lionel Messi
Luis Suarez
```

```
#include <iostream>
#include <vector>

using namespace std;

class Player{

 string name;
 string team;

public:
 Player (string name, string team)
 {
  this->name=name;
  this->team=team;
 }

  Player()
  {

   }

 string get_name()
 {
  return name;
 }

 string get_team()
 {
  return team;
 }

};
```

```
class Goal{

  Player p; //Player who scored goal-one of the attributes of goal (we previously only used variables)

public:
  Goal(Player p)
  {
    this->p=p;
  }

  Player get_player() //since Player p is private, we have a getter function to access it outside
  {
    return p;
  }

};


class Soccer_game{

  vector<Goal> all_goals; //keep track of all goals made by keeping a vector of goals

public:
  void score_goal(Goal g) //when someone scores a goal, we keep it the vector above
  {
    all_goals.push_back(g);
  }

  Goal goal_scored(int n) //give goal number, return Goal scored. (-1 to get right index)
  {
    return all_goals[n-1];
  }

};



int main(int argc, char **argv)
{

  Soccer_game s1; //start a soccer game

  //create players
  Player p1("Lionel Messi", "FC Barcelona");
  Player p2("Luis Suarez", "FC Barcelona");
  Player p3("Arthur Melo", "FC Barcelona");
  Player p4("Gareth Bale", "Real Madrid");

  //create goals scored-notice the constructor takes the player that scored the goal
```

```
  Goal g1(p1);
  Goal g2(p2);
  Goal g3(p3);

  //goals are registered in the soccer game
  s1.score_goal(g1);
  s1.score_goal(g2);
  s1.score_goal(g3);

  //see who scored second goal. Remember that every function returns something and the next function is called on that return value.  For example, goal_scored() returns a Goal object and get_player() is the function called on that Goal object (since we need a Goal object to use get_player(), which is a function in the Goal class).  The return value of the last function called (get_name()) is held in the string variable called person
  string person=s1.goal_scored(1).get_player().get_name();
  cout << person <<endl;

  //also can do ^^^ directly
 // cout << s1.goal_scored(2).get_player().get_name()<<endl;

}
```

Program 3:

Create a program of customers and workers in the local DMV. There are people working in the DMV and customers in line. If a customer complains because the line is too long (when there are two people in front of him or her), the DMV kicks them out. The workers help people in line.

Setter functions (aka modifier functions-we did getter functions last time), front and back functions, pop_back function, empty function

```
computer$ g++ dmv.cpp
computer$ ./a.out
-Are you a customer or worker?
worker
No one in line.

-Are you a customer or worker?
customer
-Please enter your name to be added to the line:
Pablo
-You are currently number 1 in line.

-Are you a customer or worker?
customer
-Please enter your name to be added to the line:
Alejandro
-You are currently number 2 in line.

-Are you a customer or worker?
customer
-Please enter your name to be added to the line:
Joaquin
-You are currently number 3 in line.
```

```
Joaquin says: This line is too long!!!! Come on!!!
-Kicking out Joaquin for complaining. >:(

-Are you a customer or worker?
worker
-Helping customer: Pablo
-Current number of people in line: 1.

-Are you a customer or worker?
customer
-Please enter your name to be added to the line:
Joaquin
-You are currently number 2 in line.

-Are you a customer or worker?
worker
-Helping customer: Alejandro
-Current number of people in line: 1.

-Are you a customer or worker?
worker
-Helping customer: Joaquin
-Current number of people in line: 0.

-Are you a customer or worker?
worker
No one in line.

-Are you a customer or worker?
exit
Dmv is closing.  Exiting system...
```

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Person{

	string name;

public:
	Person(string name) //constructor
	{
		this->name=name; //setting name with the argument passed in
	}

	string get_name() //getter function (Accessor method)
	{
		return name;
	}

	void complain()
	{
		cout<<name<<" says: This line is too long!!!! Come on!!!"<<endl;
	}

};
```

```
class Dmv{

	vector<Person> line; //people get added to line
	bool status; //open (true) or closed (false)

public:

	Dmv(bool status) //constructor
	{
		this->status=status;
	}

	//return 0 if complain, 1 if not complaining
	int add_line()
	{
		string answer;
		cout<<"-Please enter your name to be added to the line:"<<endl;
		cin>>answer;

		Person p(answer);
		line.push_back(p);
		cout<<"-You are currently number "<<line.size()<<" in line.\n"<<endl;

		if(2<line.size())
		{
			p.complain();
			return 0;
		}

		else
		{
			return 1; //not complain
		}

	}

	void help_customer()
	{

		if(line.empty())
		{
			cout<<"No one in line.\n"<<endl;
		}

		else
		{
			cout<<"-Helping customer: "<<line.front().get_name()<<endl;
			line.erase(line.begin());
```

```
                cout<<"-Current number of people in line: "<<line.size()<<".\n"<<endl;
        }


    }

    void kick_out() //kick out last person for complaining
    {
        //accessing the last person:

        cout<<"-Kicking out "<<line.back().get_name()<<" for complaining. >:(\n"<<endl; //last()
to access last person
        line.pop_back(); //gets rid of last element in vector
    }

    void set_dmv_status(bool status) //open or closed?
    {
        this->status=status;
    }//setter

    bool get_dmv_status() //getter
    {
        return status;
    }


};

int main(int argc, char **argv)
{
    int n;
    Dmv office(true);
    string answer;


    while(office.get_dmv_status())
    {
        cout<<"-Are you a customer or worker?"<<endl;
        cin>>answer;

        if(answer=="worker")
        {
            office.help_customer();
        }

        else if(answer=="customer")
        {
            n=office.add_line();

            if(n==0)
```

```
                {
                        office.kick_out();
                }
        }

        else if(answer=="exit")
        {
                cout<<"Dmv is closing.  Exiting system..."<<endl;
                office.set_dmv_status(false);
        }

        else
        {
                cout<<"-Invalid entry.\n"<<endl;
        }

    }

}
```
