Şafak AKINCI **25.11.2016**
152120141003

Experiment 4
Classes and Objects-I

Objectives

To write simple computer programs using classes and objects.

Prelab Activities

Programming Output

1-
```
account1 balance: $ 3550
```

2-
```
Error: Initial balance cannot be negative.
account1 balance: $ 0
```

3-
```
account1 balance: $ 1533
adding $253 to account1 balance
account1 balance: $ 1789
```

4-
```
account1 balance: $ 2770
adding $375 to account1 balance
account1 balance: $ 3145
```

5-
```
account1 balance: $ 799
adding -$114 to account1 balance
account1 balance: $ 685
```

CorrectTheCode

```
6-  logic and compilation error         GradeBook( string );          (in header file)
    logic error                         GradeBook
                                        GradeBook gradeBook ("Introduction to C++");
7-  compilation error                   gradeBook.setCourseName ("Advanced C++");

8-  compilation error                   string inputName;
    compilation error                   gradeBook.setCourseName (inputName);

9-  compilation error                   cout << "The grade book's course name is: " <<gradeBook.courseName()<< endl;
```

Lab Exercises

Lab Exercise 1 - Random Number Generator

```
/****************************************
 * Circle.h                             *
 ****************************************
 * IDE : Xcode                          *
 * Author : Şafak AKINCI                *
 * Experiment 4: Classes And Objects-I  *
 ****************************************/

class Circle
{
//Public members are accessible from anywhere where the object is visible.
public:
    //Constructor function of Circle class, when a new object is created, it will be called.
    Circle();
```

```cpp
    //Destructor function of Circle class, when object is removed, it will be called.
    ~Circle();

//Those are member functions. They can call from an object that is created from Circle class.

/*
 ***Private members of a class are accessible only from within other members of the same class.
    -->  That's why we create getter functions(also called accessor functions) to reach private members of an object
    -->  via these functions on condition that don't change them (private members will be returned but won't be
changed).
 */

    //Returns centerX (private member) as constant (doesn't change)
    float getCenterX() const;
    //Returns centerY (private member) as constant (doesn't change)
    float getCenterY() const;
    //Returns radius (private member) as constant (doesn't change)
    double getRadius() const;

    //Assigns given parameter (center_X) to object's private member centerX.
    void setCenterX(float center_X);
    //Assigns given parameter (center_Y) to object's private member centerY.
    void setCenterY(float center_Y);
    //Assigns given parameter (rad) to object's private member radius.
    void setRadius(double rad);

//Private members of a class are accessible only from within other members of the same class.
private:
    float centerX;
    float centerY;
    double radius;
};

/*****************************************
 * Circle.cpp                            *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 4: Classes And Objects-I   *
 *****************************************/

#include "Circle.h"             //To know function prototypes of Circle class.

//Constructor function of Circle class.
Circle::Circle()
{
}

//Destructor function of Circle class.
Circle::~Circle()
{

}

//Returns centerX (private member) as constant (doesn't change)
float Circle::getCenterX() const{
    return centerX;
}

//Returns centerY (private member) as constant (doesn't change)
float Circle::getCenterY() const{
    return centerY;
}

//Returns radius (private member) as constant (doesn't change)
double Circle::getRadius() const{
    return radius;
}

//Assigns given parameter (center_X) to object's private member centerX.
void Circle::setCenterX(float center_X){
    centerX = center_X;
}

//Assigns given parameter (center_Y) to object's private member centerY.
void Circle::setCenterY(float center_Y){
    centerY = center_Y;
}

//Assigns given parameter (rad) to object's private member radius.
void Circle::setRadius(double rad){
    radius = rad;
}



/*****************************************
 * RandomNumberGenerator.h               *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 4: Classes And Objects-I   *
 *****************************************/
```

```cpp
#include "Circle.h"             //To declare a function that is able to return Circle object (or its adrres).

class RandomNumberGenerator
{
//Public members are accessible from anywhere where the object is visible.
public:

    //Enums are used to don't define constans for each parameter.
    //  The idea is that instead of using an int to represent a set of values,
    //  a type(enum) with a restricted set of values in used instead.
    //If we didn't define enums we must define " const int ONE=1; " or " const int TWO=2 "
    // *** it can be integer value respectively 1,2,3,4 ***
    enum Precision { ONE, TWO, THREE, FOUR };
    enum CharacterType { UPPER_LETTER, LOWER_LETTER, DIGIT };

    //Constructor function of RandomNumberGenerator class, when a new object is created, it will be called.
    RandomNumberGenerator();

    //A virtual function or virtual method is a function whose behavior can be overridden
    //within an inheriting class by a function with the same signature.
    //Destructor function of RandomNumberGenerator class, when object is removed, it will be called.
    virtual ~RandomNumberGenerator();

//Those are member functions. They can call from an object that is created from RandomNumberGenerator class.

    //Generates integer number randomly between lowerBound and upperBound.
    int getRandomInteger(int lowerBound, int upperBound);

    //Generates float number randomly between lowerBound and upperBound
    //and its fractional part is defined by precision (it can be integer value respectively 1,2,3,4).
    float getRandomFloat(float lowerBound, float upperBound, Precision precision);

    //Generates double number randomly between lowerBound and upperBound
    //and its fractional part is defined by precision (it can be integer value respectively 1,2,3,4).
    double getRandomDouble(double lowerBound, double upperBound, Precision precision);

    //Generates a character randomly intended character type (it is defined by CharacterType enum).
    char getRandomCharacter(CharacterType characterType);

    //Generates a circle whose radius will be between given parameters minRadius and maxRadius
    //and its fractional part is defined by precision (it can be integer value respectively 1,2,3,4).
    Circle getRandomCircle(float minRadius, float maxRadius, Precision precision);
};
```

```
/*****************************************
 * RandomNumberGenerator.cpp            *
 *****************************************
 * IDE : Xcode                          *
 * Author : Şafak AKINCI                *
 * Experiment 4: Classes And Objects-I   *
 *****************************************/

#include "RandomNumberGenerator.h"  //To know function prototypes of RandomNumberGenerator class.
#include <time.h>                   //To use srand() function.
#include <iostream>                 //To use standart input output functions.
#include <math.h>                   //To use pow() and sqrt() functions.
using namespace std;                //To use standart input and output functions under the "std" namespace.

//Constructor Function is called when an object is created.
RandomNumberGenerator::RandomNumberGenerator()
{
    //To don't get the same random numbers.
    srand((unsigned) time (NULL));
}

RandomNumberGenerator::~RandomNumberGenerator()
{

}

//Generates integer number randomly between lowerBound and upperBound and returns it.
int RandomNumberGenerator::getRandomInteger(int lowerBound, int upperBound)
{
    int randomInteger;

    //For example, lowerBound=5, upperBound=8, To get random number between 5 and 8,
    //Firstly, get a random number from rand() function,
    //  Then, divide it to 4(upperBound-lowerBound+1).
    //  That means, random number can be 0,1,2,3 which are the possible remainder values of division.
    //      Then, add 5(lowerBound) to the remainder (0,1,2,3).
    //          Now, random number can be 5,6,7,8. (lowerBound and upperBound)
    randomInteger = lowerBound + rand()%(upperBound-lowerBound+1);

    return randomInteger;
}//end RandomNumberGenerator::getRandomInteger ()

//Generates float number randomly between lowerBound and upperBound
//and its fractional part is defined by precision (it can be integer value respectively 1,2,3,4).
float RandomNumberGenerator::getRandomFloat(float lowerBound, float upperBound, Precision precision)
{
    float randomFloat;
    int multiplied_lower_bound;
    int multiplied_upper_bound;

    //The size of the number's fractional part will be one.
    if(precision == ONE)
    {
        //Multiplies both lower and upper bound variables by pow(10,precision) and stores them different variables of
type integer.
        multiplied_lower_bound = lowerBound * pow(10, precision);
        multiplied_upper_bound = upperBound * pow(10, precision);

        //Obtains a random integer value using multiplied boundaries and stores it in a float variable.
        randomFloat = getRandomInteger(multiplied_lower_bound, multiplied_upper_bound);

        //Divides obtained float value by pow(10,precision).
        randomFloat = randomFloat / pow(10,precision);
    }
    //The size of the number's fractional part will be two.
    else if(precision == TWO)
    {
        //Multiplies both lower and upper bound variables by pow(10,precision) and stores them different variables of
type integer.
        multiplied_lower_bound = lowerBound * pow(10, precision);
        multiplied_upper_bound = upperBound * pow(10, precision);

        //Obtains a random integer value using multiplied boundaries and stores it in a float variable.
        randomFloat = getRandomInteger(multiplied_lower_bound, multiplied_upper_bound);

        //Divides obtained float value by pow(10,precision).
        randomFloat = randomFloat / pow(10, precision);
    }
    //The size of the number's fractional part will be three.
    else if(precision == THREE)
    {
        //Multiplies both lower and upper bound variables by pow(10,precision) and stores them different variables of
type integer.
        multiplied_lower_bound = lowerBound * pow(10, precision);
        multiplied_upper_bound = upperBound * pow(10, precision);

        //Obtains a random integer value using multiplied boundaries and stores it in a float variable.
        randomFloat = getRandomInteger(multiplied_lower_bound, multiplied_upper_bound);

        //Divides obtained float value by pow(10,precision).
        randomFloat = randomFloat / pow(10, precision);
    }
    //The size of the number's fractional part will be four.
```

```cpp
    else if(precision == FOUR)
    {
        //Multiplies both lower and upper bound variables by pow(10,precision) and stores them different variables of 
type integer.
        multiplied_lower_bound = lowerBound * pow(10, precision);
        multiplied_upper_bound = upperBound * pow(10, precision);

        //Obtains a random integer value using multiplied boundaries and stores it in a float variable.
        randomFloat = getRandomInteger(multiplied_lower_bound, multiplied_upper_bound);

        //Divides obtained float value by pow(10,precision).
        randomFloat = randomFloat / pow(10, precision);
    }

    return randomFloat;
}//end RandomNumberGenerator::getRandomFloat ()

//Generates double number randomly between lowerBound and upperBound
//and its fractional part is defined by precision (it can be integer value respectively 1,2,3,4).
double RandomNumberGenerator::getRandomDouble(double lowerBound, double upperBound, Precision precision)
{
    float randomDouble;

    int multiplied_lower_bound;
    int multiplied_upper_bound;

    //The size of the number's fractional part will be one.
    if(precision == ONE)
    {
        //Multiplies both lower and upper bound variables by pow(10,precision) and stores them different variables of 
type integer.
        multiplied_lower_bound = lowerBound * pow(10, precision);
        multiplied_upper_bound = upperBound * pow(10, precision);

        //Obtains a random integer value using multiplied boundaries and stores it in a float variable.
        randomDouble = getRandomInteger(multiplied_lower_bound, multiplied_upper_bound);

        //Divides obtained float value by pow(10,precision).
        randomDouble = randomDouble / pow(10,precision);
    }
    //The size of the number's fractional part will be two.
    else if(precision == TWO)
    {
        //Multiplies both lower and upper bound variables by pow(10,precision) and stores them different variables of 
type integer.
        multiplied_lower_bound = lowerBound * pow(10, precision);
        multiplied_upper_bound = upperBound * pow(10, precision);
        //Obtains a random integer value using multiplied boundaries and stores it in a float variable.
        randomDouble = getRandomInteger(multiplied_lower_bound, multiplied_upper_bound);

        //Divides obtained float value by pow(10,precision).
        randomDouble = randomDouble / pow(10, precision);
    }
    //The size of the number's fractional part will be three.
    else if(precision == THREE)
    {
        //Multiplies both lower and upper bound variables by pow(10,precision) and stores them different variables of
type integer.
        multiplied_lower_bound = lowerBound * pow(10, precision);
        multiplied_upper_bound = upperBound * pow(10, precision);

        //Obtains a random integer value using multiplied boundaries and stores it in a float variable.
        randomDouble = getRandomInteger(multiplied_lower_bound, multiplied_upper_bound);

        //Divides obtained float value by pow(10,precision).
        randomDouble = randomDouble / pow(10, precision);
    }
    //The size of the number's fractional part will be four.
    else if(precision == FOUR)
    {
        //Multiplies both lower and upper bound variables by pow(10,precision) and stores them different variables of
type integer.
        multiplied_lower_bound = lowerBound * pow(10, precision);
        multiplied_upper_bound = upperBound * pow(10, precision);

        //Obtains a random integer value using multiplied boundaries and stores it in a float variable.
        randomDouble = getRandomInteger(multiplied_lower_bound, multiplied_upper_bound);

        //Divides obtained float value by pow(10,precision).
        randomDouble = randomDouble / pow(10, precision);
    }

    return randomDouble;
}//end RandomNumberGenerator::getRandomDouble ()

//Generates a character randomly intended character type (it is defined by CharacterType enum).
char RandomNumberGenerator::getRandomCharacter(CharacterType characterType)
{
    char randomCharacter;
    //  In the ASCII table:
    //  The upper letters are between 65(A) and 90(Z)
    //  The lower letters are between 97(a) and 122(z)
    //  The digits are between 48(0) and 57(9)
    if(characterType == UPPER_LETTER)
    {
        randomCharacter =(char)getRandomInteger(65, 90);;
    }
    else if(characterType == LOWER_LETTER)
    {
        randomCharacter = (char)getRandomInteger(97, 122);
    }
    else if(characterType == DIGIT)
    {
        //  getRandomInteger(0, 9);
        randomCharacter = (char)getRandomInteger(48, 57);
    }

    return randomCharacter;
}//end RandomNumberGenerator::getRandomCharacter ()

//Generates a circle whose radius will be between given parameters minRadius and maxRadius
//and its fractional part is defined by precision (it can be integer value respectively 1,2,3,4).
Circle RandomNumberGenerator::getRandomCircle(float minRadius, float maxRadius, Precision precision)
{
    Circle circ;

    //circ.setRadius( getRandomDouble(minRadius, maxRadius, precision) );
    double random_Radius = getRandomDouble(minRadius, maxRadius, precision);
    circ.setRadius(random_Radius);

    //circ.setCenterX( getRandomFloat(minRadius, maxRadius, precision) );
    float random_Center_X = getRandomFloat(minRadius, maxRadius, precision);
    circ.setCenterX(random_Center_X);

    //circ.setCenterY( getRandomFloat(minRadius, maxRadius, precision) );
    float random_Center_Y = getRandomFloat(minRadius, maxRadius, precision);
    circ.setCenterY(random_Center_Y);

    return circ;
}//end RandomNumberGenerator::getRandomCircle ()


/*****************************************
 * RandomNumberGeneratorTestMain.cpp     *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 4: Classes And Objects-I   *
 *****************************************/
```

```cpp
#include "RandomNumberGenerator.h"      //To use functions which is declared in RandomNumberGenerator.h .
#include <iostream>                     //To use standart input output functions.
using namespace std;                    //To use standart input and output functions under the "std" namespace.

//Checks the integer number that is generated by getRandomInteger function whether it is between asked range or not.
void TEST_RandomInteger(RandomNumberGenerator& generator, int lowerBound, int upperBound)
{
    int randomNumber = generator.getRandomInteger(lowerBound, upperBound);

    if (randomNumber >= lowerBound && randomNumber <= upperBound)
    {
        cout << "SUCCESS:" << randomNumber << endl;
    }
    else
    {
        cout << "FAILURE : Obtained number is not between the range [" <<
        lowerBound << "," << upperBound << "]" << endl;
    }
}//end TEST_RandomInteger ()

//Checks the float number that is generated by getRandomFloat function whether it is between asked range or not.
void TEST_RandomFloat(RandomNumberGenerator& generator, float lowerBound, float upperBound,
RandomNumberGenerator::Precision precision)
{
    float randomNumber = generator.getRandomFloat(lowerBound, upperBound, precision);

    if (randomNumber >= lowerBound && randomNumber <= upperBound)
    {
        cout << "SUCCESS:" << randomNumber << endl;
    }
    else
    {
        cout << "FAILURE : Obtained number is not between the range [" <<
        lowerBound << "," << upperBound << "]" << endl;
    }

}//end TEST_RandomFloat ()

//Checks the double number that is generated by getRandomDouble function whether it is between asked range or not.
void TEST_RandomDouble(RandomNumberGenerator& generator, double lowerBound, double upperBound,
RandomNumberGenerator::Precision precision)
{
    double randomNumber = generator.getRandomDouble(lowerBound, upperBound, precision);

    if (randomNumber >= lowerBound && randomNumber <= upperBound)
    {
        cout << "SUCCESS:" << randomNumber << endl;
    }
    else
    {
        cout << "FAILURE : Obtained number is not between the range [" <<
        lowerBound << "," << upperBound << "]" << endl;
    }
}//end TEST_RandomDouble ()

//Gets a character that is created by getRandomCharacter function, and prints "SUCCESS".
void TEST_RandomCharacter(RandomNumberGenerator& generator, RandomNumberGenerator::CharacterType type)
{
    char randomCharacter = generator.getRandomCharacter(type);

    cout << "SUCCESS:" << randomCharacter << endl;
}//TEST_RandomCharacter ()

//Gets a circle object that is created by getRandomCircle function (RandomNumberGenerator's class function),
//and prints its private members (centerX, centerY, radius).
void TEST_RandomCircle(RandomNumberGenerator& generator, float lowerBound, float upperBound,
RandomNumberGenerator::Precision precision)
{
    Circle crc = generator.getRandomCircle(lowerBound, upperBound, precision);
    cout<<"Radius:\t"<<crc.getRadius()<<endl;
    cout<<"CenterX:\t"<<crc.getCenterX()<<endl;
    cout<<"CenterY:\t"<<crc.getCenterY()<<endl;
}

int main ()
{
    RandomNumberGenerator generator;

    cout<< "+---------------------+" << endl
        << "| Random Integer Test |" << endl
        << "+---------------------+" << endl;

    TEST_RandomInteger(generator, 5, 20);
    TEST_RandomInteger(generator, 2, 60);

    cout<< "+-------------------+" << endl
        << "| Random Float Test |" << endl
        << "+-------------------+" << endl;

    TEST_RandomFloat(generator, 5, 20, RandomNumberGenerator::Precision::ONE);
    TEST_RandomFloat(generator, 5, 20, RandomNumberGenerator::Precision::TWO);
    TEST_RandomFloat(generator, 5, 20, RandomNumberGenerator::Precision::THREE);
    TEST_RandomFloat(generator, 5, 20, RandomNumberGenerator::Precision::FOUR);
    cout<< "+--------------------+" << endl
        << "| Random Double Test |" << endl
        << "+--------------------+" << endl;

    TEST_RandomDouble(generator, 5, 20, RandomNumberGenerator::Precision::ONE);
    TEST_RandomDouble(generator, 5, 20, RandomNumberGenerator::Precision::TWO);
    TEST_RandomDouble(generator, 5, 20, RandomNumberGenerator::Precision::THREE);
    TEST_RandomDouble(generator, 5, 20, RandomNumberGenerator::Precision::FOUR);

    cout<< "+-----------------------+" << endl
        << "| Random Character Test |" << endl
        << "+-----------------------+" << endl;

    TEST_RandomCharacter(generator, RandomNumberGenerator::CharacterType::LOWER_LETTER);
    TEST_RandomCharacter(generator, RandomNumberGenerator::CharacterType::UPPER_LETTER);
    TEST_RandomCharacter(generator, RandomNumberGenerator::CharacterType::DIGIT);

    //    AT QUIZ
    cout<< "+-----------------------+" << endl
        << "|    Random Circle      |" << endl
        << "+-----------------------+" << endl;

    TEST_RandomCircle(generator, 5, 20, RandomNumberGenerator::Precision::ONE);

    return 0;

}//end main ()
```

Lab Exercise 2 - Vehicle

```
/******************************************
 * Vehicle.h                              *
 ******************************************
 * IDE : Xcode                            *
 * Author : Şafak AKINCI                  *
 * Experiment 4: Classes And Objects-I    *
 ******************************************/

/*
    In the C and C++ programming languages, " #pragma once " is a non-standard
 but widely supported preprocessor directive designed
 to cause the current source file to be included ONLY ONCE IN A SINGLE COMPILATION.
    " #pragma once " has several advantages, including:
 less code, avoidance of name clashes, and sometimes improvement in compilation speed.
```

```cpp
 *** In this case, because of the #pragma once preprocessor directive (it works for Visual Studio),
 *** Vehicle.h file will compile just for once.
 */

#pragma once
#include <iostream>                  //To use standart input output
using namespace std;                 //To use standart input and output functions under the "std" namespace.

class Vehicle
{
//Public members are accessible from anywhere where the object is visible.
public:

    /// <summary>
    /// Initializes a new instance of the <see cref="Vehicle"/> class.
    /// </summary>
    /// <param name="maxPassengerNumber">The maximum passenger number.</param>
    /// <param name="maxSpeed">The maximum speed.</param>
//Constructor function of Circle class, when a new object is created, it will be called.
    Vehicle(int maxPassengerNumber = 4, double maxSpeed = 180);

    /// <summary>
    /// </summary>
    /// <param name="amount">The amount.</param>
    /// <returns></returns>
//Acceleration of the car depends on the engine status, if engine is not started, accelerate function returns false.
    bool Accelarate(double amount);

    /// <summary>
    /// Decelerates by specified amount.
    /// </summary>
    /// <param name="amount">The amount.</param>
    /// <returns></returns>
//Deceleration of the car depends on the engine status, if engine is not started, decelerate function returns false.
    bool Decelerate(double amount);

    /*
     ***Private members of a class are accessible only from within other members of the same class.
     -->  That's why we create getter functions(also called accessor functions) to reach private members of an object
     -->  via these functions.
                +getCurrentPassengerNumber()
                +getCurrentSpeed()
                +IsEngineStarted()
     -->  (Private members will be returned but they shouldn't be changed).
     */

    /// <summary>
    /// Gets the current passenger number.
    /// </summary>
    /// <returns></returns>
//Returns currentPassengerNumber(double) private member.
//(Accessor function, it should be const to don't change class's member.)
    double getCurrentPassengerNumber();

    /// <summary>
    /// Gets the current speed.
    /// </summary>
    /// <returns></returns>
//Returns currentSpeed(double) private member.
//(Accessor function, it should be const to don't change class's member.)
    double getCurrentSpeed();

    /// <summary>
    /// Gets the in.
    /// </summary>
    /// <param name="passengerNumber">The passenger number.</param>
//GetIn function depends on the currentSpeed and passengerCount. Changes the passengerNumber. (Mutator function)
    bool GetIn(int passengerNumber);

    /// <summary>
    /// Gets the out.
    /// </summary>
    /// <param name="passengerNumber">The passenger number.</param>
    /// <returns></returns>
//GetOut function depends on the currentSpeed and passengerCount. Changes the passengerNumber. (Mutator function)
    bool GetOut(int passengerNumber);

    /// <summary>
    /// Starts the engine.
    /// </summary>
//Prints a message on the screen and set the bool member variable called engineStarted. (Mutator function)
    void StartEngine();

    /// <summary>
    /// Stops the engine.
    /// </summary>
//Prints a message on the screen and set the bool member variable called engineStarted. (Mutator function)
    void StopEngine();

    /// <summary>
    /// Determines whether [is engine started].
    /// </summary>
    /// <summary>
    /// The engine started
    /// </summary>
//Returns engineStarted(bool) private member.
//(Accessor function, it should be const to don't change class's member.)
    bool IsEngineStarted();

    /// <summary>
    /// Finalizes an instance of the <see cref="Vehicle"/> class.
    /// </summary>
//Destructor function of Vehicle class, when object is removed, it will be called.
    virtual ~Vehicle();

//Private members of a class are accessible only from within other members of the same class.
private:
    /// <summary>
    /// The current passenger number
    /// </summary>
    int currentPassengerNumber;

    /// <summary>
    /// The current speed
    /// </summary>
    double currentSpeed;

    /// <summary>
    /// The engine started
    /// </summary>
    bool engineStarted;

    /// <summary>
    /// The maximum passenger number
    /// </summary>
    int maxPassengerNumber;

    /// <summary>
    /// The maximum speed
    /// </summary>
    double maxSpeed;
};
```

```
/****************************************
 * Vehicle.cpp                          *
 ****************************************
 * IDE : Xcode                          *
 * Author : Şafak AKINCI                *
 * Experiment 4: Classes And Objects-I  *
 ****************************************/

#include "Vehicle.h"                          //To know function prototypes of Vehicle class.

//Constructor function of Vehicle class. Initializes member variables (private).
Vehicle::Vehicle(int maxPassengerNumber, double maxSpeed)
{
    this->maxSpeed = maxSpeed;
    this->maxPassengerNumber = maxPassengerNumber;
    this->engineStarted = false;
    this->currentSpeed = 0;
    this->currentPassengerNumber = 0;
    //this->airConditionDegree = 22;
}//end Vehicle ()

//Destructor function of Vehicle class.
Vehicle::~Vehicle()
{

}//end ~Vehicle ()

//Acceleration of the car depends on the engine status, if engine is not started, accelerate function returns false.
bool Vehicle::Accelarate(double amount)
{
    //Acceleration limit also depends on the maxSpeed of the vehicle.

    if(IsEngineStarted() && currentSpeed<maxSpeed)
    {
        //If any attempt to increase the speed of the vehicle to a higher value than maxSpeed,
        //just set the currentSpeed to maxSpeed value.
        if(amount>=maxSpeed || (currentSpeed+amount >= maxSpeed) )
            this->currentSpeed = maxSpeed;

        else
            this->currentSpeed = currentSpeed + amount;

        return true;
    }//end if

    return false;
}//end Vehicle::Accelarate ()
```

```cpp
//Deceleration of the car depends on the engine status, if engine is not started, decelerate function returns false.
bool Vehicle::Decelerate(double amount)
{
    //Deceleration limit also depends on the currentSpeed of the vehicle.

    if(IsEngineStarted() && currentSpeed>0)
    {
        //If any attempt to decrease the speed of the vehicle to a higher value than currentSpeed,
        //just set the currentSpeed to 0 value.
        if(amount>=currentSpeed || (currentSpeed-amount <= 0) )
            this->currentSpeed = 0;

        else
            this->currentSpeed = currentSpeed - amount;

        return true;
    }//end if

    return false;
}//end Vehicle::Decelerate ()

//Prints a message on the screen and set the bool member variable called engineStarted true.
void Vehicle::StartEngine()
{
    cout<<"Engine Started."<<endl;
    engineStarted = true;
}//end Vehicle::StartEngine ()

//Prints a message on the screen and set the bool member variable called engineStarted false.
void Vehicle::StopEngine()
{
    cout<<"Engine Stopped."<<endl;
    engineStarted = false;
}//end Vehicle::StopEngine ()

//GetIn function depends on the currentSpeed and passengerCount.
bool Vehicle::GetIn(int passengerNumber)
{

    //If current speed is non-zero value, function returns false.
    //If the currentPassenger number is not enough to GetIn, prints an error message and returns false.
    if( currentSpeed==0 && currentPassengerNumber+passengerNumber <= maxPassengerNumber )
    {
        currentPassengerNumber = currentPassengerNumber + passengerNumber;
        //        cout<<"SUCCESS : Got-In"<<endl;
        return true;
    }

    cout<<"ERROR : Cannot Get-In the vehicle while moving."<<endl;
    return false;
}//end Vehicle::GetIn ()

//GetOut function depends on the currentSpeed and passengerCount.
bool Vehicle::GetOut(int passengerNumber)
{

    //If current speed is non-zero value, function returns false.
    //If the currentPassenger number is not enough to GetOut, prints an error message and returns false.
    if( currentSpeed==0 && currentPassengerNumber != 0 && currentPassengerNumber >= passengerNumber )
    {
        currentPassengerNumber = currentPassengerNumber - passengerNumber;
        //         cout<<"SUCCESS : Got-Out"<<endl;
        return true;
    }

    cout<<"ERROR : Cannot Get-Out the vehicle while moving."<<endl;
    return false;
}//end Vehicle::GetOut ()

//Returns currentSpeed(double) private member.
//(Accessor function, it should be const to don't change class's member.)
double Vehicle::getCurrentSpeed()
{
    return currentSpeed;
}//end Vehicle::getCurrentSpeed ()

//Returns currentPassengerNumber(double) private member.
//(Accessor function, it should be const to don't change class's member.)
double Vehicle::getCurrentPassengerNumber()
{
    return currentPassengerNumber;
}//end Vehicle::getCurrentPassengerNumber ()

//Returns engineStarted(bool) private member.
//(Accessor function, it should be const to don't change class's member.)
bool Vehicle::IsEngineStarted()
{
    return engineStarted;
}//end Vehicle::IsEngineStarted ()
```

```
/****************************************
 * VehicleTestMain.cpp                  *
 ****************************************
 * IDE : Xcode                          *
 * Author : Şafak AKINCI                *
 * Experiment 4: Classes And Objects-I  *
 ****************************************/

#include <iostream>                  //To use standart input output
#include <string>                    //Was not used.
#include "Vehicle.h"                 //To use functions which are declared in Vehicle.h.
using namespace std;                 //To use standart input and output functions under the "std" namespace.

/// <summary>
/// Acceleration Test.
/// </summary>
/// <param name="vehicle">The vehicle.</param>
/// <param name="amount">The amount.</param>
    //Checks whether the vehicle accelerated or not.
void TEST_Acceleration(Vehicle& vehicle, double amount)
{
    double previousSpeed = vehicle.getCurrentSpeed();

    if (vehicle.Accelarate(amount))
    {
        cout << "SUCCESS : Accelerated " << endl;
        cout << "Previous Speed : "<< previousSpeed <<" Current Speed :  "<<
        vehicle.getCurrentSpeed() << endl;
    }

    else
    {
        cout << "FAILURE : Could not accelerated" << endl;
    }
}//end TEST_Acceleration()

/// <summary>
/// Deceleration Test.
/// </summary>
/// <param name="vehicle">The vehicle.</param>
    //Checks whether the vehicle decelerated or not.
void TEST_Deceleration(Vehicle& vehicle, double amount)
{
    double previousSpeed = vehicle.getCurrentSpeed();
```

```
        if (vehicle.Decelerate(amount))
        {
            cout << "SUCCESS : Decelerated " << endl;
            cout << "Previous Speed : "<< previousSpeed <<" Current Speed :  "<<
            vehicle.getCurrentSpeed() << endl;
        }
        else
        {
            cout << "FAILURE : Could not decelerated" << endl;
        }
    }//end TEST_Deceleration()

    /// <summary>
    /// GetIn Test.
    /// </summary>
    /// <param name="vehicle">The vehicle.</param>
    /// <param name="passengerNumber">The passenger number.</param>
        //Checks whether the passengers get in car or not.
    void TEST_GetIn(Vehicle& vehicle, int passengerNumber)
    {
        int previousPassengerNumber = vehicle.getCurrentPassengerNumber();

        if(vehicle.GetIn(passengerNumber))
        {
            cout << "SUCCESS : Got-In " << endl;
            cout << "Previous PassengerNumber : "<< previousPassengerNumber <<" Current PassengerNumber :  "<<
            vehicle.getCurrentPassengerNumber() << endl;
        }
        else
        {
            cout << "FAILURE : Could not got-in" << endl;
        }

    }//end TEST_GetIn()

    /// <summary>
    /// Get Out Test.
    /// </summary>
    /// <param name="vehicle">The vehicle.</param>
    /// <param name="passengerNumber">The passenger number.</param>
        //Checks whether the passengers get out car or not.
    void TEST_GetOut(Vehicle& vehicle, int PassengerNumber)
    {
        int previousPassengerNumber = vehicle.getCurrentPassengerNumber();

        if(vehicle.GetOut(PassengerNumber))
        {
```

```cpp
            cout << "SUCCESS : Got-In " << endl;
            cout << "Previous PassengerNumber : "<< previousPassengerNumber <<" Current PassengerNumber :  "<<
            vehicle.getCurrentPassengerNumber() << endl;
        }
        else{
            cout << "FAILURE : Could not got-out" << endl;
        }

    }//end TEST_GetOut ()

    /// <summary>
    /// Start Engine Test.
    /// </summary>
    /// <param name="vehicle">The vehicle.</param>
    void TEST_StartEngine(Vehicle& vehicle)
    {

    }//end TEST_StartEngine()

    /// <summary>
    /// Stop Engine Test.
    /// </summary>
    /// <param name="vehicle">The vehicle.</param>
    void TEST_StopEngine(Vehicle& vehicle)
    {

    }//end TEST_StopEngine

    /// <summary>
    /// Main function.
    /// </summary>
    /// <returns></returns>
    int main()
    {
        cout<< "+-----------------------+" << endl
        << "| TEST OF FIRST VEHICLE |" << endl
        << "+-----------------------+" << endl;

        Vehicle vehicle1(4, 220);
        vehicle1.StartEngine();
        TEST_Acceleration(vehicle1, 30);
        TEST_Deceleration(vehicle1, 20);
        TEST_Acceleration(vehicle1, 300);
        TEST_GetIn(vehicle1, 2);
        TEST_GetOut(vehicle1, 1);
        vehicle1.StopEngine();
        cout<< "+-------------------------+" << endl
        << "| TEST OF SECOND VEHICLE |" << endl
        << "+-------------------------+" << endl;

        Vehicle vehicle2(5, 180);
        TEST_Acceleration(vehicle2, 30);
        TEST_Deceleration(vehicle2, 20);
        TEST_GetIn(vehicle2, 2);
        TEST_GetOut(vehicle2, 1);

        return 0;
    }//end main ()
```

Conclusion

***   Classes contain functions as well as data members.**

***   Accessing those members and functions can be limited or not via access specifiers called private and public.**

***   Private members of an object can be accessed by using object's member function, we call these functions as "Accessor Functions". They are used for accessing the private member of the object but they shouldn't change the value of private members. So, they should define as const functions.**

***   Private members of an object can also be changed by the object's member functions, we call these functions as "Mutator Functions".**

***   Constructor functions which is called when a new object is created. They are generally used for initializing members.**

***   Destructor function which is called when an object is removed.**

***   Building complex objects from simpler ones is called OBJECT COMPOSITION. We've created a function(in RandomNumberGenerator Class) that returns an object(Circle) which is defined in other Class (Circle Class).**