National University
of computer and emerging sciences

Lab 7

Items	Description
Course Title	Object Oriented Programming
Lab Title	Classes
Duration	3 Hours
Tools	Eclipse/ C++
Objective	To get familiar with the use of different concepts in classes in c++

1. Classes using Header Files

In C++, it's common to split the class definition (in a `.h` file) from the class implementation (in a `.cpp` file). This provides better organization and readability, especially in larger projects.

Explanation:

- **Header File (`.h`)**: Contains the class declaration (data members and function prototypes).
- **Source File (`.cpp`)**: Contains the implementation of the class methods.

Book.h

```
#ifndef BOOK_H
#define BOOK_H

#include <string>

```

```cpp
class Book {
private:
    std::string title;
    std::string author;
    int pages;

public:
    // Constructor
    Book(const std::string, const std::string, int);

    // Getter methods
    std::string getTitle() const;
    std::string getAuthor() const;
    int getPages() const;

    // Setter methods
    void setTitle(const std::string);
    void setAuthor(const std::string);
    void setPages(int p);
};

#endif
```

National University
of computer and emerging sciences

Book.cpp

```cpp
#include "book.h"

// Constructor implementation

Book::Book(const std::string t, const std::string a, int
p){

    title = t;

    author = a;

    pages = p;

}

// Getter methods implementation

std::string Book::getTitle() const { return title; }

std::string Book::getAuthor() const { return author; }

int Book::getPages() const { return pages; }

// Setter methods implementation

void Book::setTitle(const std::string t) { title = t; }

void Book::setAuthor(const std::string a) { author = a; }

void Book::setPages(int p) { pages = p; }
```

Main.cpp:

```
#include <iostream>
#include "book.h"

int main() {
    // Creating a Book object
    Book book("C++ Primer", "Stanley Lippman", 1024);

    // Accessing the Book details
    std::cout << "Title: " << book.getTitle() <<
std::endl;
    std::cout << "Author: " << book.getAuthor() <<
std::endl;
    std::cout << "Pages: " << book.getPages() <<
std::endl;

    return 0;
}
```

How to execute:

g++ book.cpp main.cpp -o out.exe

2. Const Objects

A `const` object in C++ ensures that the object's state cannot be modified. This is useful when you want to prevent accidental changes to the object.

```
#include <iostream>

class Book
{
public:
    std::string title;

    Book(const std::string t)
    {
        title = t;
    }

    // Const member function (can be called on const
objects)
    std::string getTitle() const
    {
        return title;
    }

    // Non-const member function (cannot be called on
const objects)
    void setTitle(const std::string t)
    {
```

```cpp
        title = t;
    }
};

int main()
{
    const Book myBook("Effective C++");

    std::cout << "Title: " << myBook.getTitle() <<
std::endl;
    // The following line would result in a compiler
error:
    // myBook.setTitle("New Title");
    return 0;
}
```

Key Points:

- **Const objects** can only call **const member functions**.
- Non-const member functions cannot modify the state of a const object.

3. Copy Constructor

A **copy constructor** creates a new object as a copy of an existing object. If you don't provide one, the compiler generates a default copy constructor, which performs a shallow copy. You can define your own copy constructor to implement a deep copy.

```cpp
#include <iostream>
using namespace std;

class Book {
public:
    string* title;


    Book(const string t) {
        title = new string(t);
    }


    ~Book() {
        delete title;
    }


    // Custom copy constructor
    Book(const Book& other) {
        title = other.title;
    }
};

int main() {
    Book book1("C++ Programming");
    Book book2 = book1; // Calls the copy constructor
    *book2.title = "Python Programming"; // Only affects
book2
    cout << "Book1 Title: " << *book1.title << endl; //
Outputs: C++ Programming
    cout << "Book2 Title: " << *book2.title << endl; //
Outputs: Python Programming

    return 0;
}
```

4. Shallow Copy vs Deep Copy

When copying objects, C++ can create either a **shallow copy** or a **deep copy**.

- **Shallow Copy**: The copied object shares the same memory (pointer) as the original object, leading to potential issues if one object modifies the memory.
- **Deep Copy**: A new copy of the data is made, so the copied object is completely independent of the original.

Shallow Copy:

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

class Book {
public:
    string* title;


    Book(const string t) {
        title = new string(t);
    }


    ~Book() {
        delete title;
    }


    // Custom copy constructor
    Book(const Book& other) {
        title = other.title;
    }
};

int main() {
    Book book1("C++ Programming");
    Book book2 = book1; // Calls the copy constructor


    *book2.title = "Python Programming"; // Only affects
book2
    cout << "Book1 Title: " << *book1.title << endl; //
Outputs: C++ Programming
    cout << "Book2 Title: " << *book2.title << endl; //
Outputs: Python Programming

    return 0;
}
```

Deep Copy:

```cpp
#include<iostream>
using namespace std;

class Book {
public:
    string* title;

    Book(const string t) {
        title = new string(t);
    }

    ~Book() {
        delete title;
    }

    // Deep copy constructor
    Book(const Book& other) {
        title = new string(*other.title); // Allocate new memory
    }
};

int main() {
    Book book1("C++ Programming");
    Book book2 = book1; // Deep copy

    *book2.title = "Python Programming"; // Only affects book2
    cout << "Book1 Title: " << *book1.title << endl; //
Outputs: C++ Programming
    cout << "Book2 Title: " << *book2.title << endl; //
Outputs: Python Programming

    return 0;
}
```

5. Static Data Members and Member Functions

- **Static Data Members**: Shared by all objects of the class. There is only one copy of the static member, regardless of how many objects are created.
- **Static Member Functions**: Can be called without creating an object of the class. These functions can only access static data members.

```
#include <iostream>

using namespace std;

class Book {

public:

    string title;

    static int bookCount; // Static data member

    Book(const string t) {

        title = t;

        bookCount++; // Increment static member
    }

    // Static member function

    static int getBookCount() {

        return bookCount;

    }

};

// Initialize static data member

int Book::bookCount = 0;

int main() {

    Book book1("C++ Programming");

    Book book2("Python Programming");

    // Accessing static member function

    cout << "Number of books: " << Book::getBookCount()
<< endl;

    return 0;

}
```

Key Points:

- **Static members** belong to the class, not to any specific object.
- **Static functions** can access only static members of the class.

National University of computer and emerging sciences

LAB TASKS

For all the tasks use header file for class definition.

Task 1:

Write a C++ program that uses the concept of static data members and functions to implement a class representing a bank account. The class should have the following functionalities:

1. The ability to set an initial balance for the account.
2. The ability to deposit and withdraw money from the account, with appropriate error checking to ensure that the balance does not become negative.
3. The ability to retrieve the current balance of the account.
4. The ability to retrieve the total number of bank accounts that have been created.

void deposit(double amount)
void withdraw(double amount)
double getBalance() const
static int getNumAccounts()

Define the static data member and static function as appropriate within the class.

Task 2:

Suppose you are tasked with designing a class to represent a car rental agency. Each car in the rental agency has a unique ID number, a make and model, a rental rate per day, and a current rental status (available or rented). The rental agency wants you to create a class that can be used to manage their fleet of rental cars. The class should have the following features:

1. A constructor that takes in the make and model of the car and initializes its rental rate per day to $50 and its rental status to "available". The constructor should also assign a unique ID number to each car, starting from 1 and incrementing by 1 for each new car added. **Hint**: You can use the static data member
2. A member function named "rent" that takes in the number of days a customer wants to rent the car for and updates the rental status of the car to "rented". The function should also calculate and return the total rental cost based on the rental rate per day and the number of days rented. **int rent(int num_days)**
3. A member function named "return_car" that updates the rental status of the car to "available" and calculates and returns the rental fee owed by the customer based on the number of days rented and the rental rate per day. If the car has not been rented, the function should return 0. **int return_car(int num_days)**

4. A static data member that keeps track of the total number of cars in the rental agency's fleet.
5. A static function named "get_num_cars" returns the total number of cars in the rental agency's fleet. **static int get_num_cars()**

   Your task is to implement the Car Rental class in C++, including the constructor, member functions, and static data member/function. Generate a proper menu for the selection of choices and display messages where required. E.g “The car status is already Rented”.

Task 3:

You are tasked with designing an inventory management system for a library. Each **Book** has the following attributes:

- A dynamically allocated string for the **title**.
- An integer **ID** that represents a unique identifier for the book.
- An integer **stock**, which represents the number of copies available in the library.

Your task is to:

1. Implement a **copy constructor** for the `Book` class.
2. Demonstrate **shallow copy** behavior.
3. Write a function `updateStock()` that modifies the stock of a book.
4. In `main()`, create a `Book` object and make a shallow copy. Then update the stock of the shallow copy and show how it affects the original book.

Requirements:

- **shallow copy**: Both copies should share the same memory for the `title`, so changes to one book’s `title` will affect the other.
- **updateStock()**: This method should increment or decrement the stock, demonstrating how changes in stock in one object (the copy) affect the original.

Hints:

- Think about when both objects (the original and the copy) should reference the same title, but changes to stock only affect the copy.

National University
of computer and emerging sciences

Task 4:

Design a `BankAccount` class that represents a user's bank account. Each `BankAccount` has:

- A dynamically allocated string for the **accountHolderName**.
- A dynamically allocated array for the **transaction history** (the array holds integers representing amounts deposited or withdrawn).
- An integer **balance** that shows the total balance in the account.

Your task is to:

1. Implement a **deep copy constructor** for the `BankAccount` class.
2. Demonstrate **deep copy** behavior.
3. Write a function `addTransaction(int amount)` that adds a transaction to the history and updates the balance accordingly.
4. In `main()`, create a `BankAccount` object, add some transactions, and make a deep copy of it. Then modify the transaction history of the deep copy and show that the original is not affected.

Requirements:

- **deep copy**: When making a copy of a `BankAccount`, the new object should have its own independent transaction history. Modifications to the transaction history in the copied account should not affect the original account.
- The `transaction history` should be dynamically managed (using `new[]` and `delete[]`).

Hints:

- You may need to dynamically resize the array of transactions when adding new transactions.
- Ensure that both the name and transaction history are fully independent in the copied object.