

ODD SEMESTER 2023

OOPS Lab

**Submitted for
Masters of Computer Applications**



Submitted to:

Dr. Shahab Saquib Sohail

Deptt. of Computer Science & Engg.
Jamia Hamdard, New Delhi, India

Submitted by:

Sukaina Inam Naqvi

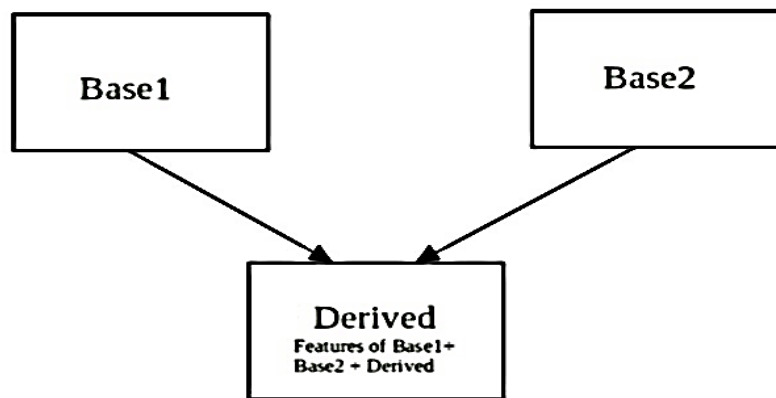
Application No: JH23-PGP-02037
MCA 1st Semester, 1st Year

Department Of Computer Science and Engineering
School of Engineering Sciences and Technology
Jamia Hamdard, New Delhi -110062

Question 1: What do you mean by multiple inheritance?

Answer: Multiple inheritance is a feature of some object-oriented computer programming languages in which an object or class can inherit features from more than one parent object or parent class. It is distinct from single inheritance, where an object or class may only inherit from one particular object or class.

A diagram that demonstrates multiple inheritance is given below –



Question 2. Why do we use inheritance?

Answer: Inheritance allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors (realizing an interface), to reuse code and to independently extend original software via public classes and interfaces.

We use inheritance in C++ for the reusability of code from the existing class. C++ strongly supports the concept of reusability. Reusability is yet another essential feature of OOP (Object Oriented Programming).

It is always good to reuse something that already exists rather than trying to create the one that is already present, as it saves time and increases reliability.

We use inheritance in C++ when both the classes in the program have the same logical domain and when we want the class to use the properties of its superclass along with its properties.

For example, there is a base class or parent class named “Animal,” and there is a child class named “Dog,” so, here dog is an animal, so in “Dog class,” all the common properties of the “Animal” class should be there, along with its property of dog animal.

Question 3. Write a program in C++ that illustrate the use of inheritance and its advantage.

Code:

```
#include <iostream>

// Base class 1
class Engine {
public:
    Engine() {
        std::cout << "Engine constructor" << std::endl;
    }
    void start() {
        std::cout << "Engine started" << std::endl;
    }
    void stop() {
        std::cout << "Engine stopped" << std::endl;
    }
};

// Base class 2
class Wheels {
public:
    Wheels() {
        std::cout << "Wheels constructor" << std::endl;
    }
    void rotate() {
        std::cout << "Wheels rotating" << std::endl;
    }
}
```

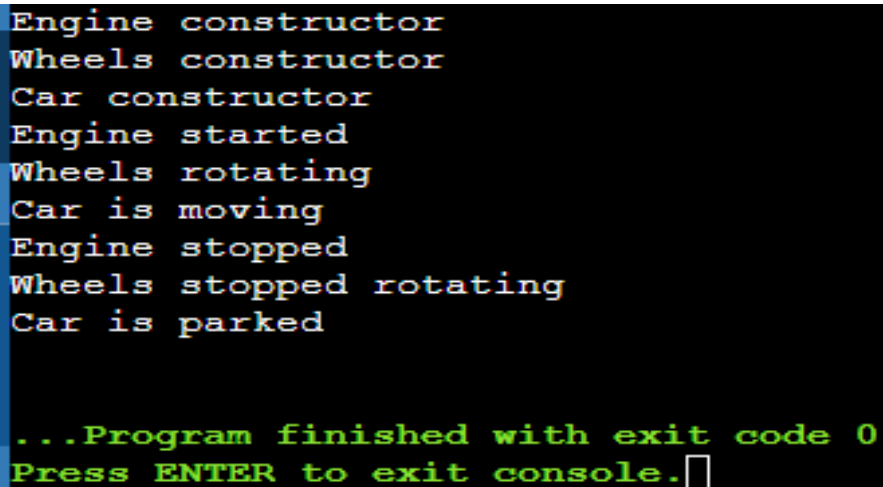
```
void stopRotation() {
    std::cout << "Wheels stopped rotating" << std::endl;
}
};

// Derived class inheriting from both Engine and Wheels
class Car : public Engine, public Wheels {
public:
    Car() {
        std::cout << "Car constructor" << std::endl;
    }
    void drive() {
        start();    // From Engine
        rotate();   // From Wheels
        std::cout << "Car is moving" << std::endl;
    }
    void park() {
        stop();     // From Engine
        stopRotation(); // From Wheels
        std::cout << "Car is parked" << std::endl;
    }
};

int main() {
    // Creating an object of the derived class
    Car myCar;
    // Accessing members from base classes
```

```
myCar.drive(); // Uses members from both Engine and Wheels  
myCar.park(); // Uses members from both Engine and Wheels  
return 0;  
}
```

Output:-



```
Engine constructor  
Wheels constructor  
Car constructor  
Engine started  
Wheels rotating  
Car is moving  
Engine stopped  
Wheels stopped rotating  
Car is parked  
  
...Program finished with exit code 0  
Press ENTER to exit console. □
```

Advantages of Inheritance in C++

Inheritance promotes reusability. When a class inherits or derives another class, it can access all the functionality of inherited class.

Reusability enhanced reliability. The base class code will be already tested and debugged.

As the existing code is reused, it leads to less development and maintenance costs.

Inheritance makes the sub classes follow a standard interface.

Inheritance helps to reduce code redundancy and supports code extensibility.

Inheritance facilitates creation of class libraries.

Question 4. Perform same task without using any OOP concept(inheritance).

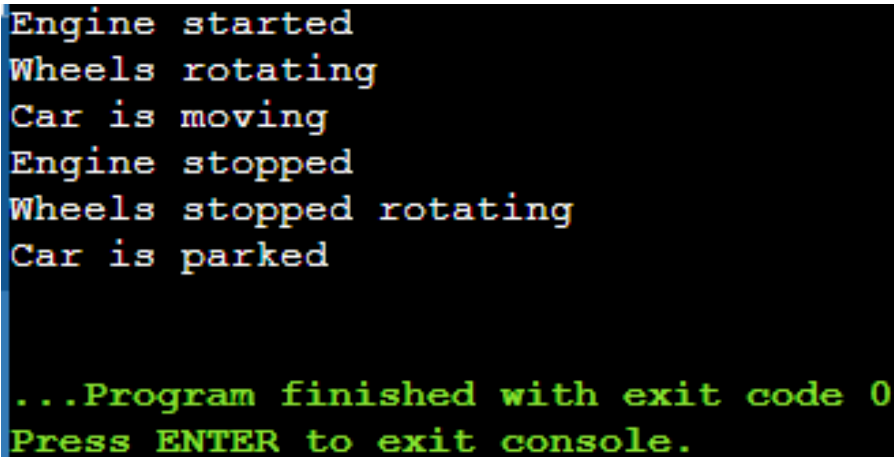
Code:

```
#include <iostream>  
  
// Engine functions
```

```
void startEngine() {  
    std::cout << "Engine started" << std::endl;  
}  
void stopEngine() {  
    std::cout << "Engine stopped" << std::endl;  
}  
// Wheels functions  
void rotateWheels() {  
    std::cout << "Wheels rotating" << std::endl;  
}  
void stopWheelRotation() {  
    std::cout << "Wheels stopped rotating" << std::endl;  
}  
// Car-related functions  
void driveCar() {  
    startEngine();  
    rotateWheels();  
    std::cout << "Car is moving" << std::endl;  
}  
void parkCar() {  
    stopEngine();  
    stopWheelRotation();  
    std::cout << "Car is parked" << std::endl;  
}  
int main() {
```

```
// Perform the same task without using inheritance  
driveCar();  
parkCar();  
return 0;  
}
```

Output:



```
Engine started  
Wheels rotating  
Car is moving  
Engine stopped  
Wheels stopped rotating  
Car is parked  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Question 5. Show how using inheritance is having advantage over procedural approach.

Answer: The length of the programs developed using OOP language is much larger than the procedural approach. Since the program becomes larger in size, it requires more time to be executed that leads to slower execution of the program.

We cannot apply OOP everywhere as it is not a universal language. It is applied only when it is required. It is not suitable for all types of problems.

Programmers need to have brilliant designing skill and programming skill along with proper planning because using OOP is little bit tricky.

OOPs take time to get used to it. The thought process involved in object-oriented programming may not be natural for some people.

Everything is treated as object in OOP so before applying it we need to have excellent thinking in terms of objects.

Question 6: Difference between Top-Down Approach and Bottom-Up Approach?

Answer: In object-oriented programming (OOP), the concepts of "top-down" and "bottom-up" can be applied to the design and implementation of software systems. A "top-down" approach in OOP involves starting with a high-level view of the system, often defining the overall architecture, class hierarchies, and major functionalities before delving into the details of individual classes and methods. Design patterns, inheritance, and abstraction play significant roles in the top-down approach, allowing developers to create a well-structured and organized system from the outset.

On the other hand, a "bottom-up" approach in OOP involves building the system from specific components or classes, gradually combining them to form higher-level structures. Developers using the bottom-up approach might begin by implementing individual classes with specific functionalities and then composing them to create more complex objects or systems. This approach emphasizes code reusability and modular design, allowing for greater flexibility and adaptability.

The main difference between the top-down and bottom-up approaches is the process's starting point and focus. The top-down approach prioritizes high-level planning and decision-making, while the bottom-up approach prioritizes the execution of individual tasks and the development of detailed knowledge. Both approaches have advantages and disadvantages, and the best approach will depend on the specific context, including the nature of the problem, the resources available, the timeline, and the desired outcome.

Question 7. It is stated that OOP is a bottom-up approach, justify the statement.

Answer: It's because you first define a class and its functions. Then you initialize an object of that class and calls functions as per the need. Now, though process looks like to-bottom but the execution takes place in bottom-up approach. While executing, execution flow finds object initialization first and then it looks up for declared class and then functions.