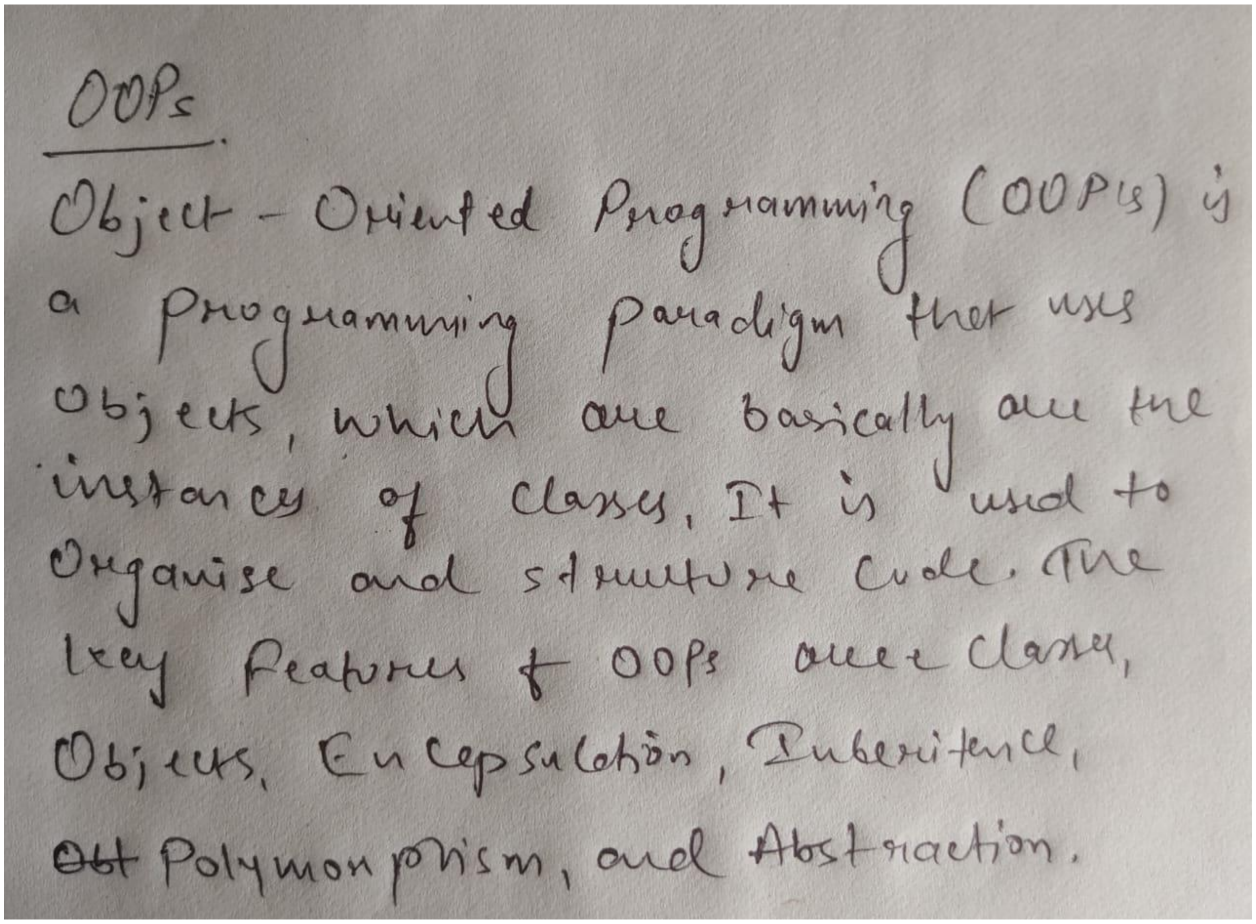
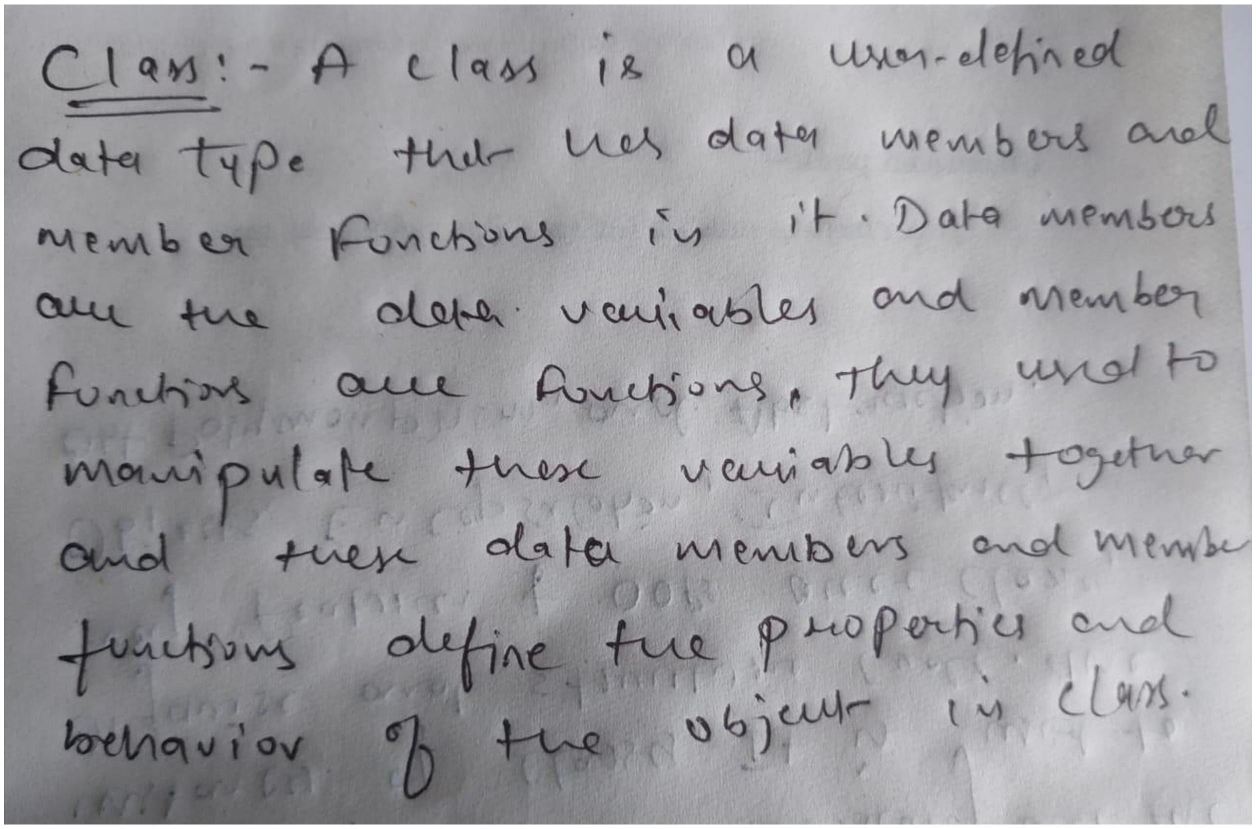
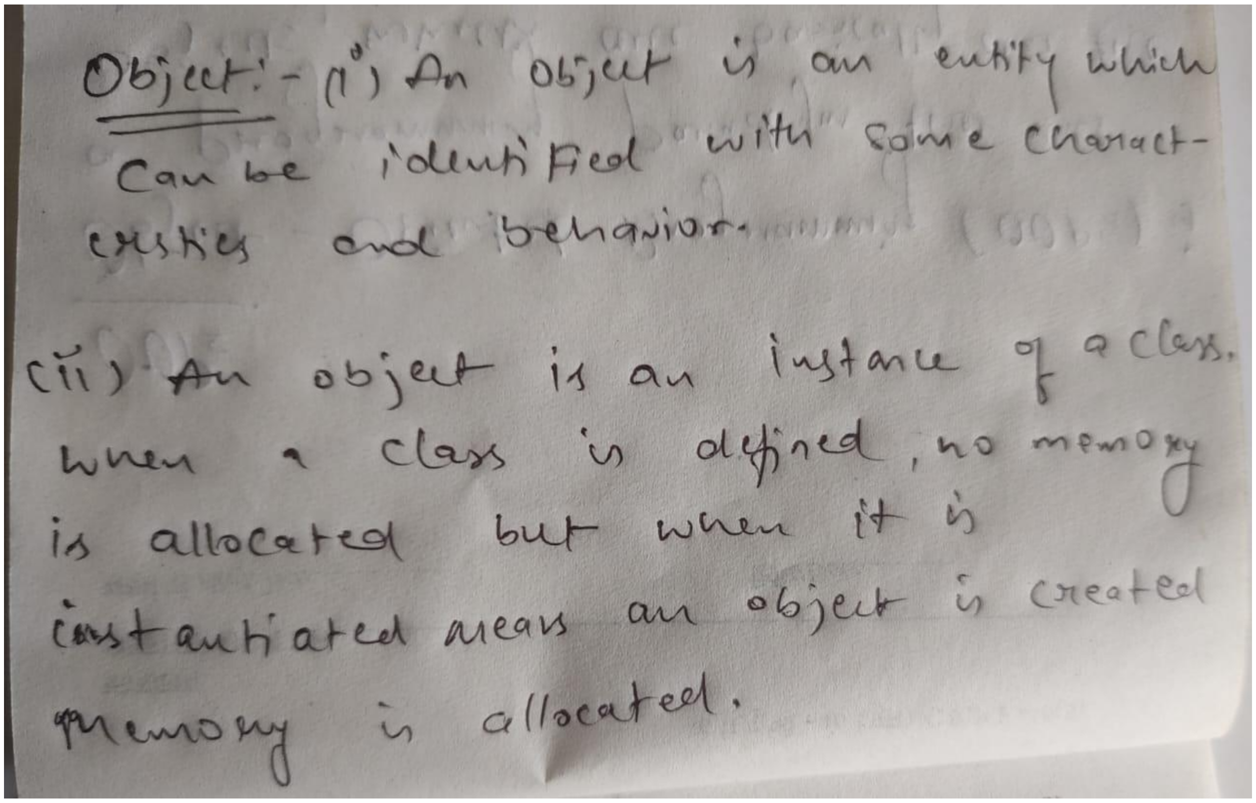


OOPs

Object - Oriented Programming (OOPs) is a programming paradigm that uses objects, which are basically are the instances of classes. It is used to Organise and structure code. The key features of OOPs are: Classes, Objects, Encepsulation, Inheritence, ~~Obt~~ Polymorphism, and Abstraction.

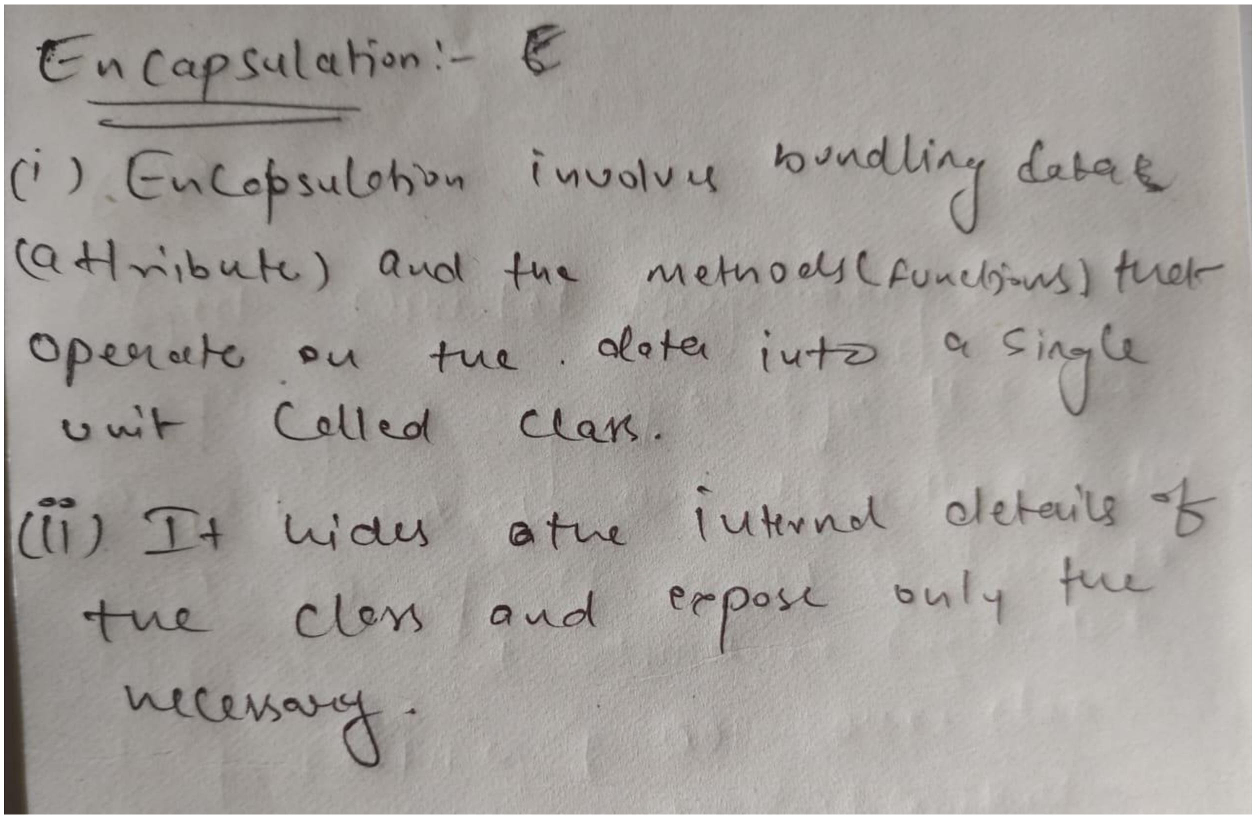


Class:- A class is a user-defined data type that uses data members and member functions in it. Data members are the data variables and member functions are functions. They used to manipulate these variables together and these data members and member functions define the properties and behavior of the object in class.



Object:- (i) An object is an entity which can be identified with some characteristics and behavior.

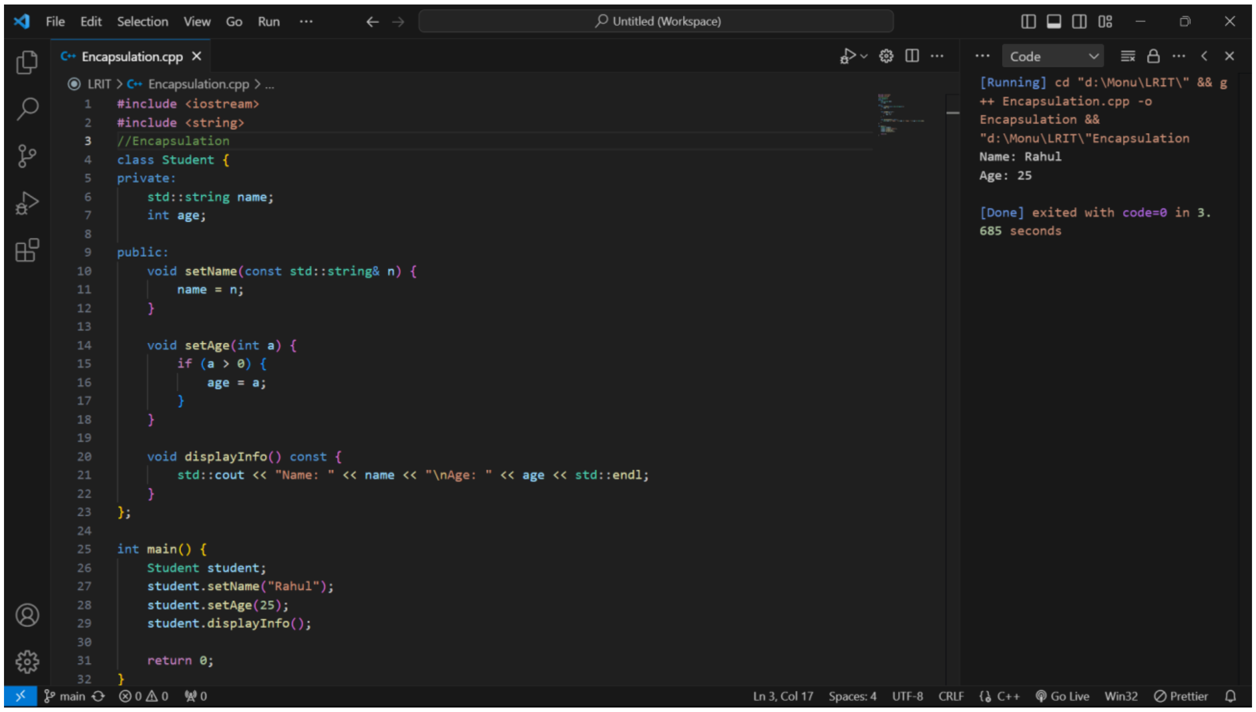
(ii) An object is an instance of a class. When a class is defined, no memory is allocated but when it is instantiated means an object is created memory is allocated.



Encapsulation :- E

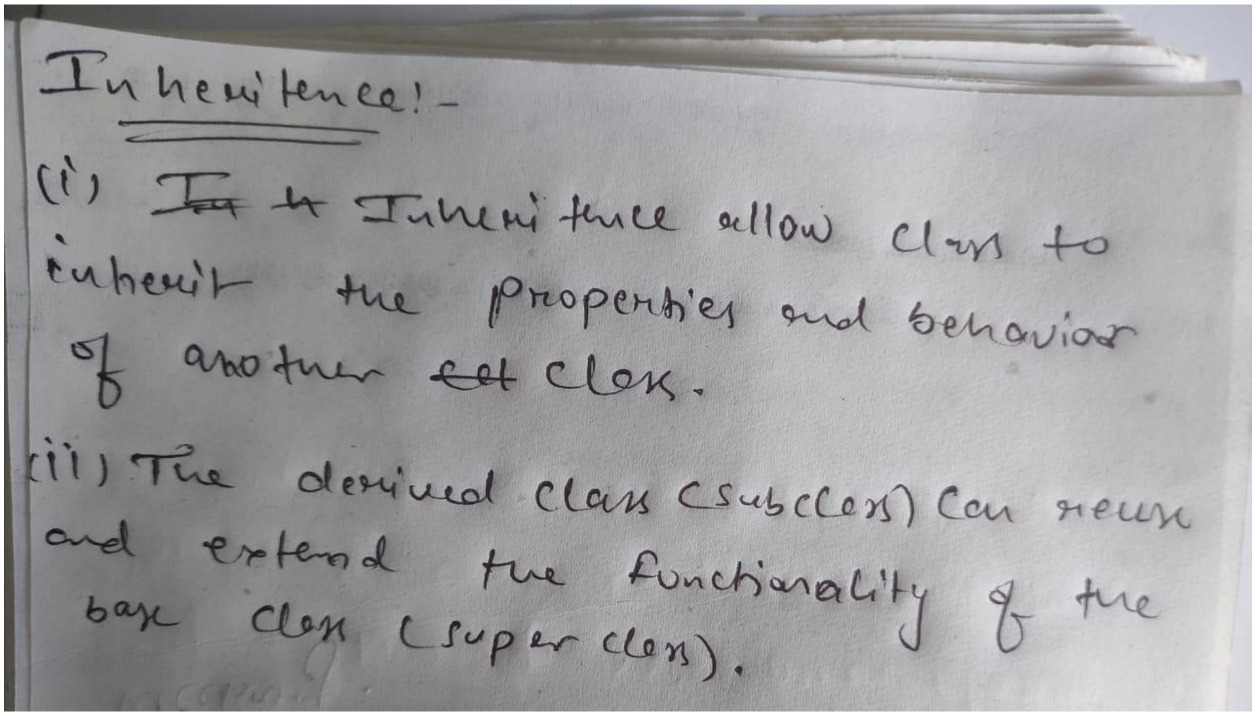
(i) Encapsulation involve bundling data & (attribute) and the methods (functions) that operate on the data into a single unit called class.

(ii) It hides athe internal details of the class and expose only the necessary.



```cpp
#include <iostream>
#include <string>
//Encapsulation
class Student {
private:
    std::string name;
    int age;

public:
    void setName(const std::string& n) {
        name = n;
    }

    void setAge(int a) {
        if (a > 0) {
            age = a;
        }
    }

    void displayInfo() const {
        std::cout << "Name: " << name << "\nAge: " << age << std::endl;
    }
};

int main() {
    Student student;
    student.setName("Rahul");
    student.setAge(25);
    student.displayInfo();

    return 0;
}
```

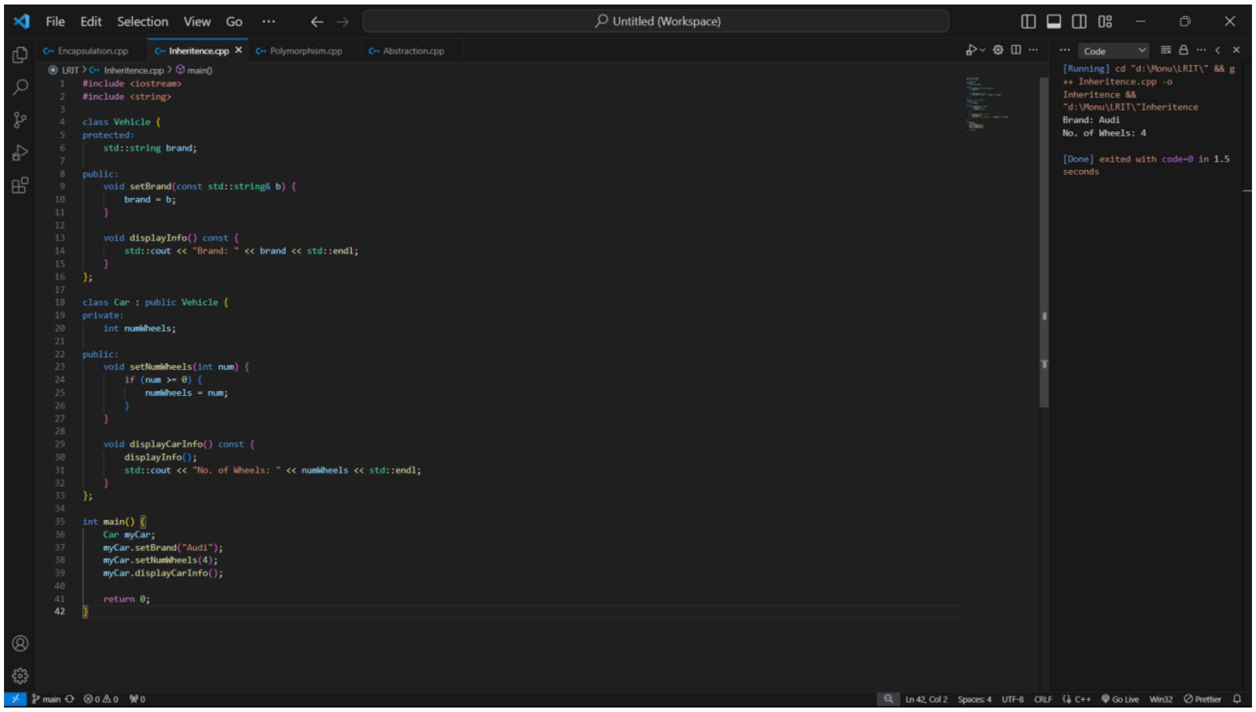
```
[Running] cd "d:\Monu\LRIT\" && g++ Encapsulation.cpp -o Encapsulation && "d:\Monu\LRIT\"Encapsulation
Name: Rahul
Age: 25

[Done] exited with code=0 in 3.685 seconds
```



Inheritence:-

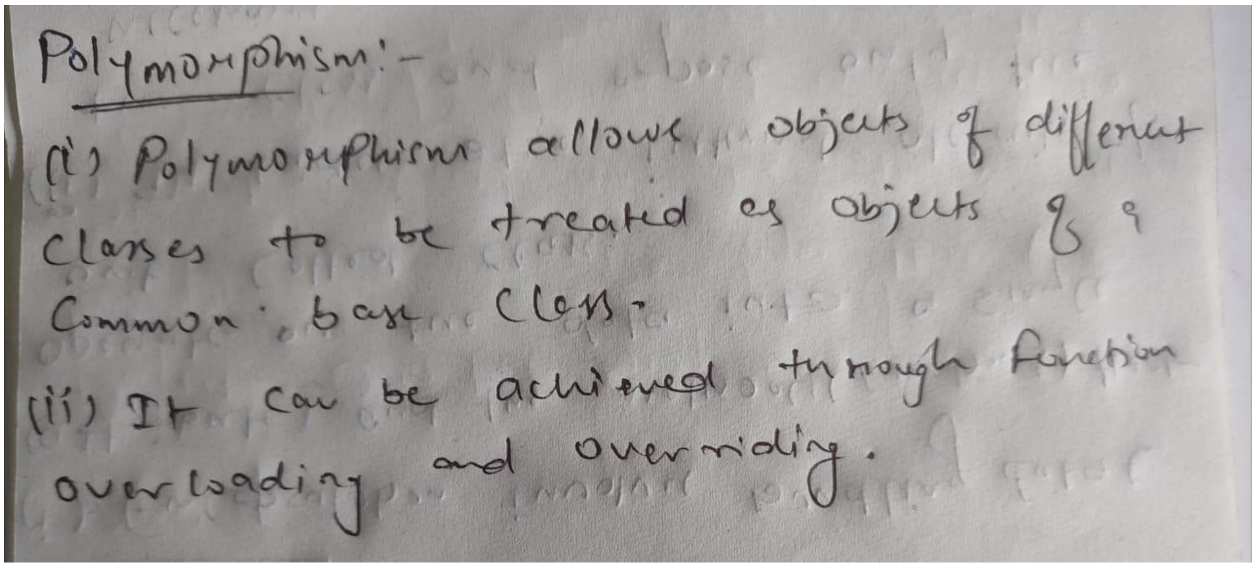
(i) ~~It~~ ~~It~~ Inheritance allow class to inherit the properties and behavior of another ~~cl~~ class.

(ii) The derived class (subclass) can reuse and extend the functionality of the base class (super class).



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

class Vehicle {
protected:
    std::string brand;

public:
    void setBrand(const std::string& b) {
        brand = b;
    }

    void displayInfo() const {
        std::cout << "Brand: " << brand << std::endl;
    }
};

class Car : public Vehicle {
private:
    int numWheels;

public:
    void setNumWheels(int num) {
        if (num >= 0) {
            numWheels = num;
        }
    }

    void displayCarInfo() const {
        displayInfo();
        std::cout << "No. of Wheels: " << numWheels << std::endl;
    }
};

int main() {
    Car myCar;
    myCar.setBrand("Audi");
    myCar.setNumWheels(4);
    myCar.displayCarInfo();

    return 0;
}
```

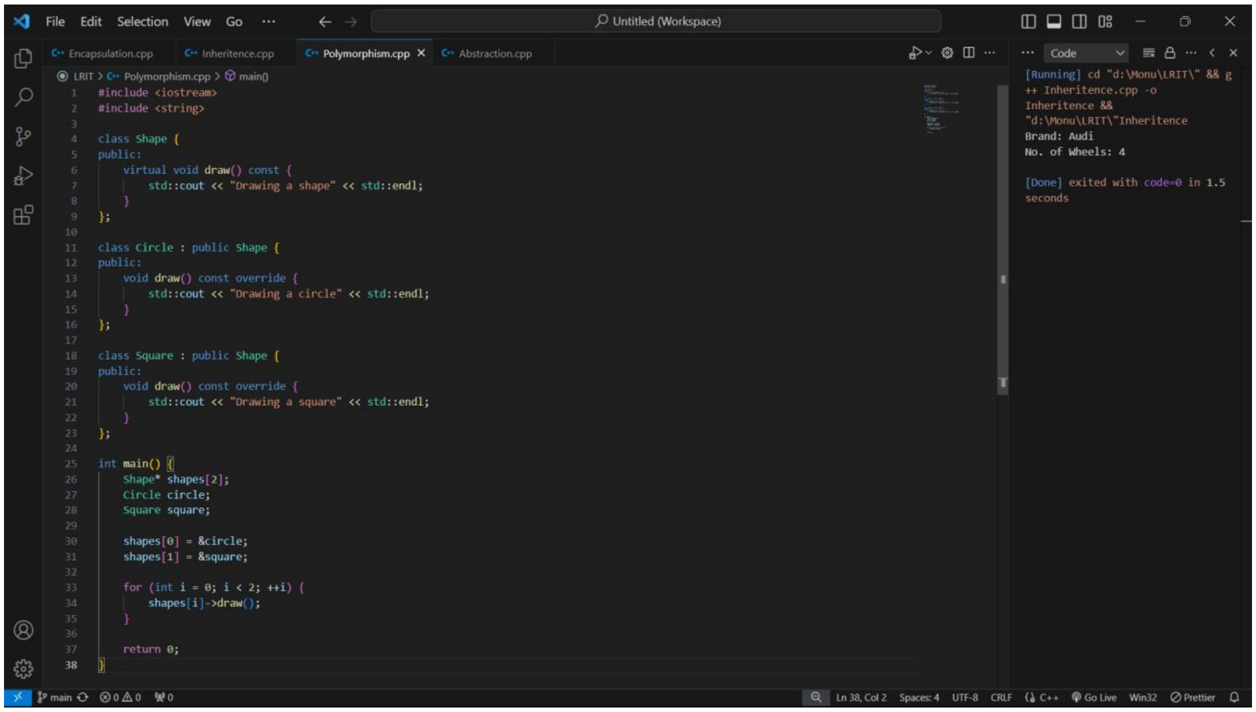
```
[Running] cd "d:\Monu\LRIT\" && g
++ Inheritence.cpp -o
Inheritence &&
"d:\Monu\LRIT\"Inheritence
Brand: Audi
No. of Wheels: 4

[Done] exited with code=0 in 1.5
seconds
```



Polymorphism:-

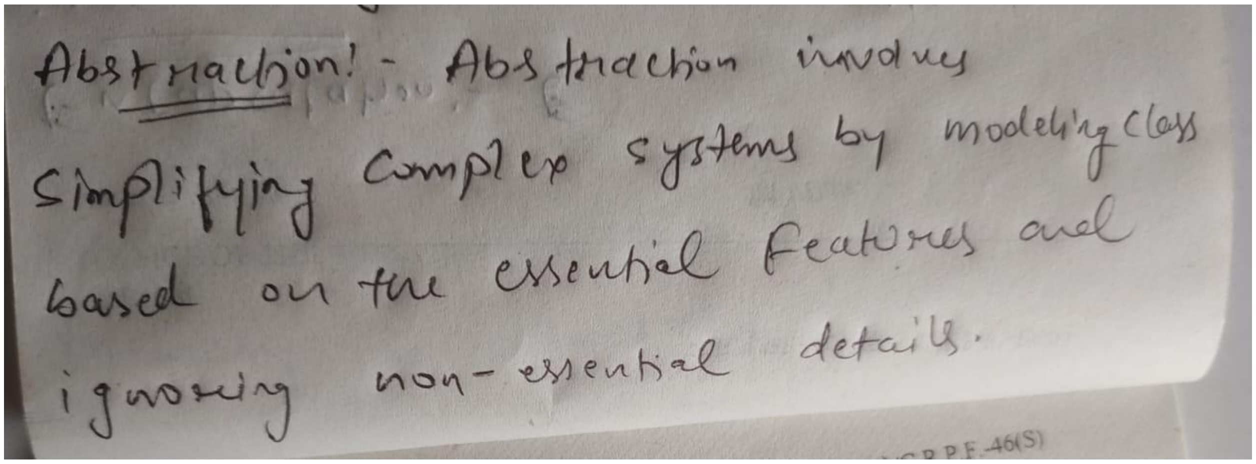
(i) Polymorphism allows objects of different classes to be treated as objects of a common base class.

(ii) It can be achieved through function overloading and overriding.



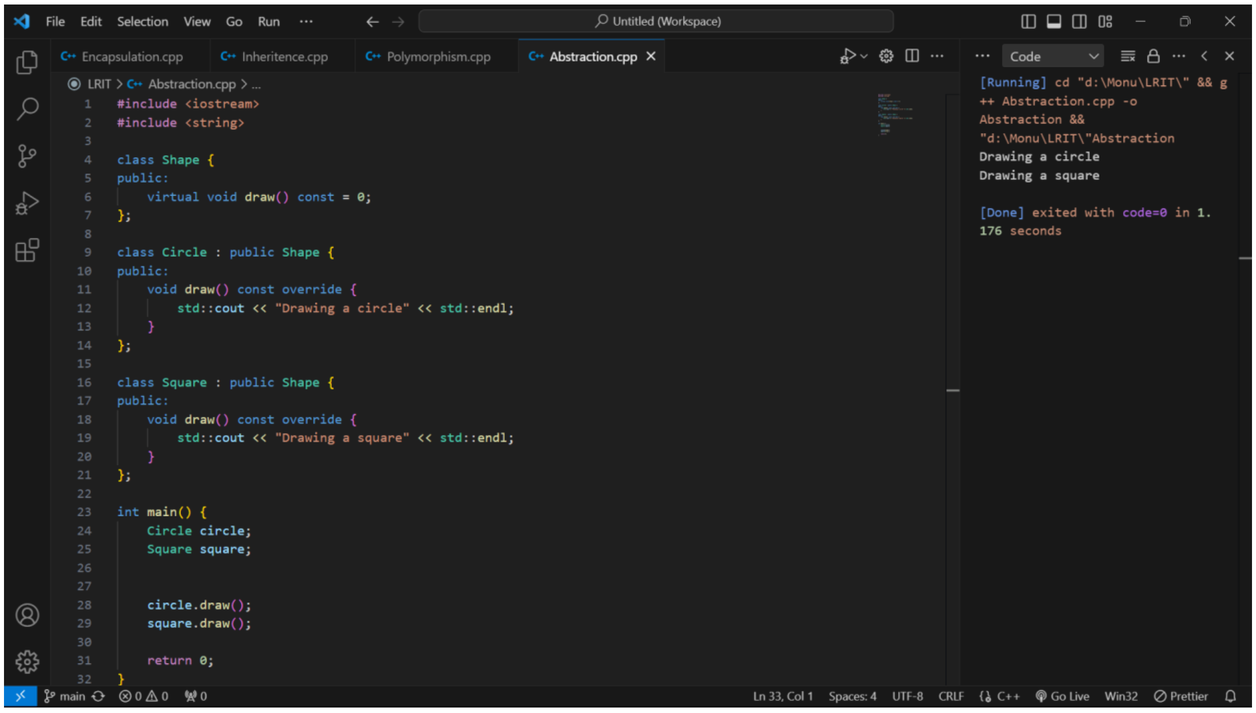
```cpp
#include <iostream>
#include <string>

class Shape {
public:
    virtual void draw() const {
        std::cout << "Drawing a shape" << std::endl;
    }
};

class Circle : public Shape {
public:
    void draw() const override {
        std::cout << "Drawing a circle" << std::endl;
    }
};

class Square : public Shape {
public:
    void draw() const override {
        std::cout << "Drawing a square" << std::endl;
    }
};

int main() {
    Shape* shapes[2];
    Circle circle;
    Square square;

    shapes[0] = &circle;
    shapes[1] = &square;

    for (int i = 0; i < 2; ++i) {
        shapes[i]->draw();
    }

    return 0;
}
```

```
[Running] cd "d:\Monu\LRIT\" && g
++ Inheritence.cpp -o
Inheritence &&
"d:\Monu\LRIT\"Inheritence
Brand: Audi
No. of Wheels: 4

[Done] exited with code=0 in 1.5
seconds
```



Abstraction! - Abstraction involves simplifying Complex systems by modeling class based on the essential Features and ignoring non-essential details.

C R P F-46(S)



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

class Shape {
public:
    virtual void draw() const = 0;
};

class Circle : public Shape {
public:
    void draw() const override {
        std::cout << "Drawing a circle" << std::endl;
    }
};

class Square : public Shape {
public:
    void draw() const override {
        std::cout << "Drawing a square" << std::endl;
    }
};

int main() {
    Circle circle;
    Square square;


    circle.draw();
    square.draw();

    return 0;
}
```

```
[Running] cd "d:\Monu\LRIT\" && g
++ Abstraction.cpp -o
Abstraction &&
"d:\Monu\LRIT\"Abstraction
Drawing a circle
Drawing a square

[Done] exited with code=0 in 1.
176 seconds
```