Q No: 1

Research the histor and development of object-oriented design. Explain its advantages....?

History:

Object-oriented design developed during the 1960s and 1970s, emerging from the creation of Simula (the first language with classes) and later languages like Smalltalk, which introduced objects and message-passing between them. By organizing code around "objects" — each representing real-world entities with distinct data (attributes) and functions (method). It provided a new way to structure programs beyond the step by step approach of procedural programming.

تاریخ: / /

M T W T F S

Procedural Programming:

It is a programming paradigm based on concept of structured, step-by-step instruction executed sequentially to perform tasks.

- sequence and structure
- function and procedures
- Global and local data

Advantage of OOD over procedural Programming

- **Modularity:** In OOD, each part of a program is encapsuled withingn object, making the code more modular and easier to maintain.

- **Reusability:** Classes and object can be reused across different projects or applications, saving time and reducing error. Through inheritance

new classes can extend existing one. For example a base "Vechile" class in a transportation app can be used extended by "Car" and "Bicycle" classes that share core attributes.

- **Scalability:** OOD supports scalable code structure that can grow as needs expand. complex application like social media platforms (Facebook) benefit from OOD

- **Abstraction:** OOD allow the developer to define complex structure in a simplified manner by using Abstraction.

- **Support for design pttern:** OOD support many desing pattern, such as singleton, Factory, and observer pattern, which provide solution to

common design problem.

Real world Application Example

They are given as:

-> Banking System:

Banking system benefit from OOD to manage a ~~vast~~ by modelling customer accounts, and transaction as separate objects, which can be updated and managed independently.

-> E-commerce Platform:

These programms like Amazon, leverage OOD to manage a vast array of object eg("Product", "User",) that interact enabling straightforward updates and new features like wish list or reviews.

QNo: 2

Design a class named Cars to stimulate a car's place basic attributes.....?

Ans:

To implement a basic simulation of a cars attributes and behaviour, lets create a 'Car' class. It will include basic attribute like speed, fuel level and acceleration.

```
#include <iostream>
#include <string>
using namespace std;
Class Car {
  Private:
    string model;
    double speed;
    double fuelLevel;
    double accelerationRate;
  Public:
// constructor to initialize car attribute
```

تاریخ: ___/___/___ M T W T F S

```
Car (string m, doubl s, doubl f, double a):
model(m), speed(s), fuelLevel(f), accelerationRate(a){
    }
// method to accelerate car
void accelerate( ){
 if (fuelLevel > 0){
    speed += accelerationRate;
    fuelLevel -= 0.5;
cout << model << "accelerated. Current speed:"
    << speed << "km/h, Fuel Level:" << fuelLevel
    << "liters." << endl;
    } else {
    cout << model << "cannot accelerate. Fuel
    Level is too low." << endl;
    }
    }
// Method to Refuel the car
void refuel(double amount){
    fuelLevel += amount;
    cout << model << "refueled. Current fuel
    Level:" << fuelLevel << "liters." << endl;
    }
```

```
// Display car current status
void displayStatus() const {
cout<< "Model: "<<model<< ", speed: "<<speed
<< " Km/h, Fuel Level: "<<fuelLevel<< " liters,
Acceleration Rate: "<< accelerationRate <<
" Km/h^2 " << endl; }
};
int main(){

Car car1("Toyota Corolla", 0, 50, 10);
Car car2("Honda Civic", 0, 60, 12);
Car car3("Ford Mustang", 0, 40, 15);
// Display initial status of car
car1.displayStatus();
car2.displayStatus();
car3.displayStatus();

cout << "\n--- Simulating Car Action ---\n" << endl;
car1.accelerate();
car2.accelerate();
car3.accelerate();
// Refuel car 1 and display status
car1.refuel(10);
```

```
// Display final state of each car
cout<<"\n--- Final status of car ---\n"<<endl;
    car1.displayStatus();
    car2.displayStatus();
    car3.displayStatus();
    return 0;
}
```

Explanation:

There are three main part of program:

1. Class Attributes: These are model, speed, fuelLevel and accelerationRate.

2. Method:

accclerate(): increased speed by accelerationRate and reduce fuel level by small ammount.

refuel(): increase fuel Level.

displayStatus(): Show the current state of the car.

Output

This programm simulate the action of accelerating, refuelling and displaying status for each car, demonstrating encapsulation in the 'Car' class.

Q No 3

Explain the concept of data encapsulation and demonstrate it by designing a bankAccount...?

Concept of Data Encapsulation

Data Encapsulation is a fundamental principle of object-oriented Programming (OOP) that restricts direct access to some of an object's components, which can prevent the accidental modification of data. This is typically achieved through the use of access modifier where certain attributes of class are private and can only be accessed through public method (functions) of that class.

It help in maintaining the integrity of the data.

```
#include <iostream>
#include <string>
Class BankAccount {
Private:
    string accontNumber;
    double balance;
Public:
 BankAccount (const string& accNumber,
    double initialBalance = 0.0) :
accountNumber(accNumber), balance(initialBalance){
        }
// Method to deposit Money
 void deposit(double amount){
  if (amount > 0) {
   balance += amount;
cout << "Deposited: $" << amount << "New
balance: $" << balance << endl;
  } else {
cout << "Deposit amount must be
  positive " << endl; }
      }
// method to withdraw money
 void withdraw(double amount){
    if(amount > 0) {
```

```
if(amount <= balance){
    balance -= amount;
cout << "Withdraw: $" << amount << ". New
balance: $" << balance << endl;
    } else {
cout << "Insufficient funds" << endl;
    }
} else {
    cout << "Withdrawal amount must
    be positive." << endl;
    } }
// Method to check current balance
double checkBalance() const {
    return balance; }
};
int main(){

BankAccount account("123456789", 1000.0);
  // check balance
cout << "Current balance: $" << account.checkBalance()
  << endl;
  // Deposit money
  account.deposit(500.0);
```

```
// Withdraw money
    account.withdraw(200.0);
    account.withdraw(1500.0);
// check balance again
cout << "Current balance: $" << amount.checkBalance();
    return 0;
}
```

Explanation:

Private members: This class has two private member accountNumber & balance

Constructor: It initialize the account number and the initial balance.

Deposit Method: The deposit method check if the amount is positive before adding to current balance.

Withdraw method: It also check that the fund is sufficient or not which is to be withdrawl.

CheckbalanceMethod: It show current balance in the account.

Output:

1: Current balance: $1000

2: Deposited: $500. New balance: $1500

3: Withdrew: $200. New balance: $1300

4: Insufficient fund.

5: Current balance: $1300

تاریخ: ___/___/___ M T W T F S

Q No 4

Design a file handler class that open file in its constructor and close in destrutor...?

Ans:

Constructor and destructor play a vital role in file management, or managing resources like file handle, network connection etc. Here is a desired program:

```
#include <iostream>
#include <fstream>
#include <string>
Class FileHandler{
Private
Public:
  // constructor that open file
FileHandler(const string& filename){
  file.open(filename);
  if(!file.is_open()){
  throw runtime_error("Failed to
  open file:" + filename);
```

```
    }
    cout << "File opened: " << filename << endl;
    }
// Destructor that close file.
~FileHandler( ){
if( file.is_open( )){
    file close ( );
cout<< "File closed." << endl;
    } }
Private:
    fstream file;
};

int main( ) {
    try {
    FileHandler fh("example.txt");
    fh.writeLine("Hello, world!");
    string line = fh.readLine( );
cout << "Read Line: " << line << endl;
    }
    catch ( const exception & e){
    cerr << "Error: " << e.what()
    << endl;
    }
```

```
    return 0;
}
```

Output:

```
File opened. example.txt
File closed
Read Line: Hello, World!
```

Importance of Constructor & Destructor in Resource Management

- **Automatic Resource Management**

constructor and destructor provide a mechanism to automatically mange resources. When an object is created, resource can be acquired and when it is destroyed, thos resources can be released

- **Exception Safety:**

If an exception is thrown during the construction of an object, the destructor will not be called, but if the object

is created successfully, the destructor will be called, but if the object goes out of scope,

▪ Encapsulation:

By encapsulating resource management within class, the complexity of resources handling is hidden from the user, making the code easier to use and less error-prone.

▪ Consistency:

Using constructor and destructor ensure that resource management is consistent throughout the codebase, which is particularly important in larger application.

Q No 5

Create a class whose name LibraryBook with attribute such as title and author.... ?

[Implementation of LibraryBook class

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

 Class LibraryBook {
 Public:
 LibraryBook (const string& newTitle
 , const string& author)
 : title (title), author (author) {
              }
 // Non const member functio
 void setTitle (const string& newTitle) {
     title = newTitle;
          }
```

```
// const member function
string getTitle() const {
    return title;
        }
// Non const member function
string getAuthor() const string& newAuthor) {
    author = new Author;
        }
// const member function
string getAuthor() const {
    return author;
        }
Private:
  string title;
  string author;
};
int main() {
LibraryBook book("1984", "George Orwell");
// Accessing non-const member function
cout<<" Original Title: "<<book.getTitle()<<endl;
cout<<" Original author: "<<book.getAuthor()<<endl;

// Modifying the title and author
```

```
book.setTitle("Animal Farm");
book.setAuthor("George Orwell");

// Accessing const member function
cout << "Updated Title: " << book.getTitle() << endl;
cout << "Updated author: " << book.getAuthor() << endl;

// Creating a const object

const LibraryBook constBook("Brave New World",
    "Aldous Huxley");
cout << "Const Book Title: " << constBook.getTitle();
cout << " Const Book Author: " << constBook.getAuthor();

return 0;
}
```

When & Why Const Member Function are useful

- **Gurantee Non-modification:**
  Marking a member function as const gurantee that it will not be modify the object state.

This is usefull for function that only need to read.

- **Const-correctnes:**

Using const member function help maintain const-correctness in your code. It allow you to use const object and ensure that API's are clear about which function can modify the object

- **Enabling Const Object**

Const member function can be called on const object which is essential for working with API's and libraries.

Output:

```
Original Title: 1984
Original Author: George Orwell
Updated Title: Animal Farm
Updated Author: George Orwell
Const Book Title: Brave New World
Const Book Author: Aldous Huxley
```