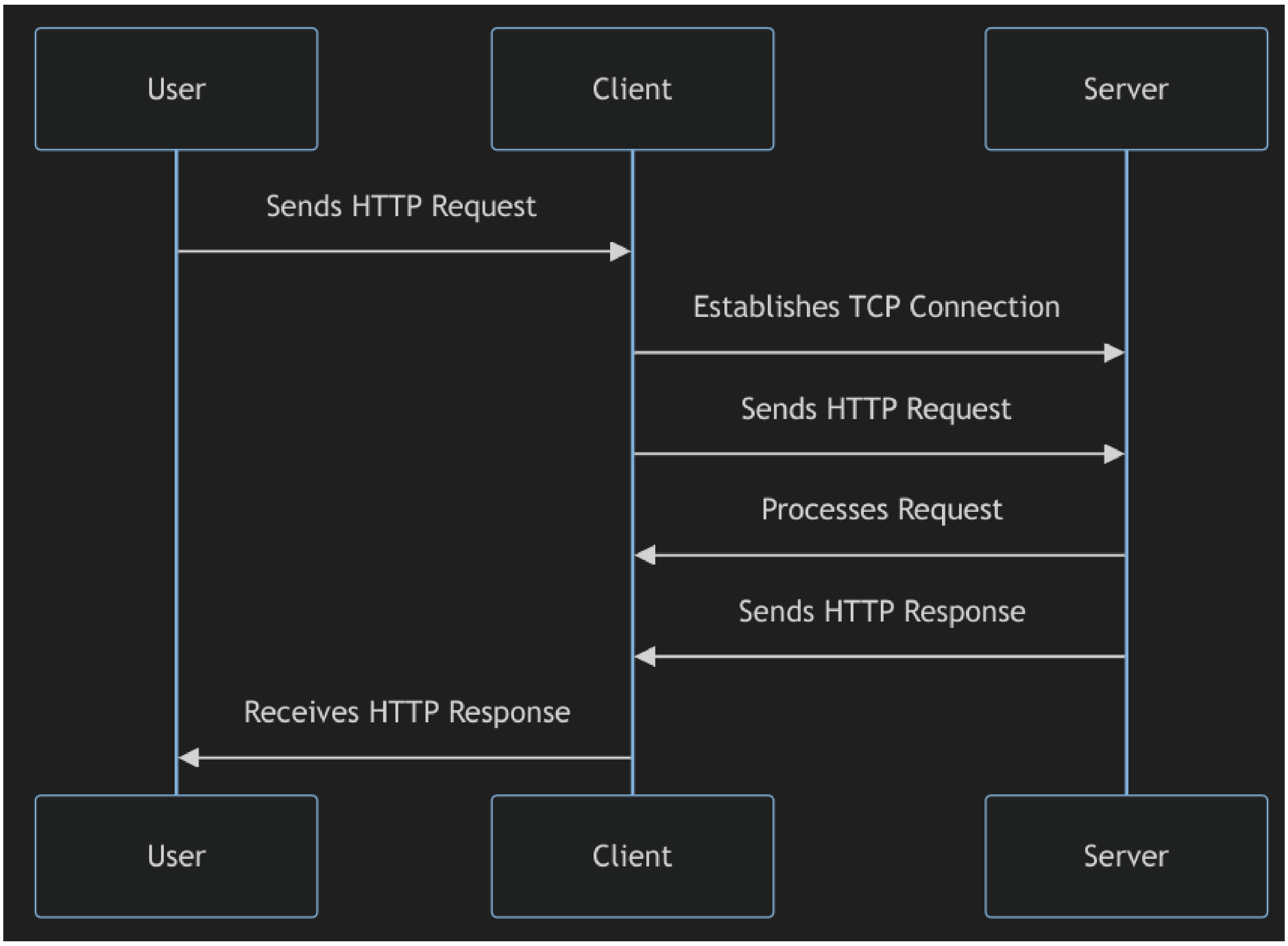
Ex9

```
#include <iostream>
#include <cstring>
#include <vector>
#include <sstream>
using namespace std;

// Ex1
class Point2D
{
private:
    double x;
    double y;

public:
    Point2D() : x(0.0), y(0.0) {}

    Point2D(double xCoord, double yCoord) : x(xCoord), y(yCoord) {}

    Point2D(const Point2D &other) : x(other.x), y(other.y) {}

    void display() const
    {
        cout << "Point coordinates = (" << x << ", " << y << ")\n";
    }
};

// Ex2
class StockManager
{
private:
    int totalavi;

public:
    StockManager() : totalavi(0) {}

    void addItem(int consumed)
    {
        totalavi += consumed;
    }

    void consumeItem(int consumed)
    {
        totalavi -= consumed;
    }

    void displayStock() const
    {
        cout << "Total items in stock = " << totalavi << endl;
    }

    void displayStock(const string &category, int consumedQuantity) const
    {
        cout << "Category = " << category << endl
             << "Consumed quantity = " << consumedQuantity << endl
             << "Remaining stock = " << (totalavi - consumedQuantity) << endl;
    }
};

// Ex3
class Person
{
private:
    string name;
    int age;
    string city;

public:
    Person()
    {
        name = "Unknown";
        age = 0;
        city = "Unknown";
    }

    Person(string n, int a)
    {
        name = n;
        age = a;
        city = "Unknown";
    }

    Person(string n, int a, string c)
    {
        name = n;
        age = a;
        city = c;
    }

    void display()
    {
        cout << "Name = " << name << endl
             << "Age = " << age << endl
             << "City = " << city << endl;
    }
};

// Ex4
class StringWrapper
{
private:
    char *str;
```

```
public:
    StringWrapper()
    {
        str = nullptr;
    }

    StringWrapper(const char *s)
    {
        if (s != nullptr)
        {
            str = new char[strlen(s) + 1];
            strcpy(str, s);
        }
        else
        {
            str = nullptr;
        }
    }

    StringWrapper(const StringWrapper &other)
    {
        if (other.str != nullptr)
        {
            str = new char[strlen(other.str) + 1];
            strcpy(str, other.str);
        }
        else
        {
            str = nullptr;
        }
    }

    ~StringWrapper()
    {
        delete[] str;
    }

    void display() const
    {
        if (str != nullptr)
        {
            cout << "String = " << str << endl;
        }
        else
        {
            cout << "String is empty" << endl;
        }
    }
};

// Ex5
class MyClass
{
private:
```

```cpp
    int value;

public:
    MyClass()
    {
        value = 0;
    }

    MyClass(int v)
    {
        value = v;
    }

    MyClass(const MyClass &obj)
    {
        value = obj.value;
    }

    void display()
    {
        cout << "Value = " << value << endl;
    }
};

// Ex6
class DynamicArray
{
private:
    int *arr;
    int size;

public:
    DynamicArray()
    {
        arr = nullptr;
        size = 0;
    }

    DynamicArray(int s)
    {
        size = s;
        arr = new int[size];
    }

    ~DynamicArray()
    {
        delete[] arr;
    }

    void display()
    {
        cout << "Array elements = ";
        for (int i = 0; i < size; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

// Ex7
class Matrix
{
private:
    vector<vector<int>> data;
    int rows, cols;

public:
    Matrix(int r, int c) : rows(r), cols(c)
    {
        data.resize(rows, vector<int>(cols, 0));
    }

    // Method to set the value of a specific element in the matrix
    void set(int row, int col, int value)
    {
        if (row >= 0 && row < rows && col >= 0 && col < cols)
        {
            data[row][col] = value;
        }
        else
        {
            cout << "Invalid index!" << endl;
        }
    }

    Matrix operator+(const Matrix &other)
    {
        if (rows != other.rows || cols != other.cols)
        {
            cout << "Matrices must have the same dimensions for addition!" <<
endl;
            return *this;
        }

        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                result.data[i][j] = data[i][j] + other.data[i][j];
            }
        }
        return result;
    }

    Matrix operator-(const Matrix &other)
    {
        if (rows != other.rows || cols != other.cols)
        {
            cout << "Matrices must have the same dimensions for subtraction!" <<
endl;
            return *this;
        }

        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                result.data[i][j] = data[i][j] - other.data[i][j];
            }
        }
        return result;
    }

    void display()
    {
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                cout << data[i][j] << " ";
            }
            cout << endl;
        }
    }
};

// Ex8
class Time
{
private:
    int hours;
    int minutes;
    int seconds;

public:
    Time(int h, int m, int s) : hours(h), minutes(m), seconds(s) {}

    Time add(Time t2)
    {
        int totalSeconds = seconds + t2.seconds;
        int totalMinutes = minutes + t2.minutes + totalSeconds / 60;
        int totalHours = hours + t2.hours + totalMinutes / 60;

        totalSeconds %= 60;
        totalMinutes %= 60;
        totalHours %= 24;

        return Time(totalHours, totalMinutes, totalSeconds);
    }
    string toString()
    {
        stringstream ss;
        ss << (hours < 10 ? "0" : "") << hours << ":" << (minutes < 10 ? "0" : "")
<< minutes << ":" << (seconds < 10 ? "0" : "") << seconds;
        return ss.str();
    }
};

int main()
{
    // Ex1
    Point2D defaultPoint;
    cout << "Default Point:\n";
    defaultPoint.display();

    Point2D pointWithCoords(3.5, 2.0);
    cout << "\nPoint with Coordinates:\n";
    pointWithCoords.display();

    Point2D copiedPoint = pointWithCoords;
    cout << "\nCopied Point:\n";
    copiedPoint.display();

    // Ex2
    StockManager stock;
    stock.addItem(100);
    stock.displayStock();
    stock.consumeItem(20);
    stock.displayStock("Food", 20);

    // Ex3
    Person p1;
    Person p2("John", 30);
    Person p3("Alice", 25, "New York");

    cout << "Person 1:" << endl;
    p1.display();

    cout << "Person 2:" << endl;
    p2.display();

    cout << "Person 3:" << endl;
    p3.display();

    // Ex4
    StringWrapper sw1;
    StringWrapper sw2("Hello");
    StringWrapper sw3 = sw2;

    cout << "String 1:" << endl;
    sw1.display();
    cout << "String 2:" << endl;
    sw2.display();

    cout << "String 3:" << endl;
    sw3.display();

    // Ex5
    MyClass obj1(10);
    MyClass obj2 = obj1;

    cout << "Object 1:" << endl;
    obj1.display();

    cout << "Object 2:" << endl;
    obj2.display();

    // Ex6
    DynamicArray arr1;
    DynamicArray arr2(5);

    arr1.display();
    arr2.display();

    // Ex7
    Matrix A(2, 3);
    A.set(0, 0, 1);
    A.set(0, 1, 2);
    A.set(0, 2, 3);
    A.set(1, 0, 4);
    A.set(1, 1, 5);
    A.set(1, 2, 6);

    Matrix B(2, 3);
    B.set(0, 0, 7);
    B.set(0, 1, 8);
    B.set(0, 2, 9);
    B.set(1, 0, 10);
    B.set(1, 1, 11);
    B.set(1, 2, 12);

    cout << "Matrix A:" << endl;
    A.display();
    cout << endl;

    cout << "Matrix B:" << endl;
    B.display();
    cout << endl;

    Matrix C = A + B;
    Matrix D = A - B;

    cout << "Matrix A + B:" << endl;
    C.display();
    cout << endl;
    cout << "Matrix A - B:" << endl;
    D.display();
    cout << endl;

    // Ex8
    int h1, m1, s1;
    int h2, m2, s2;

    cout << "Enter time 1 (hh:mm:ss): ";
    scanf("%d:%d:%d", &h1, &m1, &s1);

    cout << "Enter time 2 (hh:mm:ss): ";
    scanf("%d:%d:%d", &h2, &m2, &s2);

    Time time1(h1, m1, s1);
    Time time2(h2, m2, s2);

    Time sum = time1.add(time2);

    cout << "Sum of times: " << sum.toString() << endl;

    return 0;
}
```

Ex9


User
Client
Server
Sends HTTP Request
Establishes TCP Connection
Sends HTTP Request
Processes Request
Sends HTTP Response
Receives HTTP Response
User
Client
Server