```cpp
/*
    A)
    The DES model is implemented using a tick/clock based simulation where we use bernouli trials to approximate a poisson distribution.
    Each event (bike arrival, client(class1,2,3)) has its own random number generator. The interval selected for the bernouli approximation
    is 100000 per 1 poisson interval, meaning that for 1 poisson time interval we run 100000 bernouli trials. The reason is to make
    the approximation more accurate. We start off the simulation with X[0] = 10, and the totalMoney at (K1*r1 + K2*r2). When a bike arrives
    we increment the total number of bikes, and then distribute to anyone waiting. When a new client arrives, if a bike is available we give
    a bike to the client, otherwise we put the client onto the queue and apply the penalty (0 for class 3). We replicate the simulation
    100 times, namely because it takes such a long time to run, and then we averaged the totalMoney at the end of each run.
*/

#include <iostream>
#include <random>
#include <queue>
#include <time.h>

struct Client
{
    int type;
};

int main()
{
    const int T = 120;
    int X[121] = { 0 }; //There are T+1 events
    const double bikeArrivalRate = 6;
    //clients have rate r1 = 3, r2 = 1, r3 = 4
    const double clientRates[4] = { 0, 3.0, 1.0, 4.0 };

    //client class 1/2 pay annually (K1 = 0.5, k2 = .1), total amount is (K1*r1 + K2*r2)
    //class 3 pays per ride amount k3 = 1.25

    //when annual members (class 1/2) arrive at empty station, there is penalty c1 = 1.0, c2 = 0.25, c3 = 0
    const double clientPenalty[4] = { 0, -1.0, -0.25, 0 };

    //create and seed the generator
    std::default_random_engine generator;
    generator.seed(time(0));

    //We can use a bernouli distribution with paramter p = lambda / bernouliInterval
    int bernouliInterval = 100000;
    std::bernoulli_distribution  randomVariableGenerator[4];
```

```cpp
    randomVariableGenerator[0] = std::bernoulli_distribution(bikeArrivalRate / bernouliInterval);
    randomVariableGenerator[1] = std::bernoulli_distribution(clientRates[1] / bernouliInterval);
    randomVariableGenerator[2] = std::bernoulli_distribution(clientRates[2] / bernouliInterval);
    randomVariableGenerator[3] = std::bernoulli_distribution(clientRates[3] / bernouliInterval);

    const int numberOfTrials = 100;
    double averageMoneyAmount = 0;

    std::cout << "Starting the trials" << std::endl;

    for (int t = 0; t < numberOfTrials; t++)
    {
        //client queue
        std::queue<Client> line;

        //we can assume total money starts at 0 + the deterministic annual prorated charge of clients classes 1 and 2
        double totalMoney = (0.5 * clientRates[1]) + (0.1 * clientRates[2]);
        X[0] = 10; //we start with 10 bikes at X(0)

        for (int i = 1; i <= T; i++)
        {
            X[i] = X[i - 1]; //new time interval starts with bike amount from prev interval

            for (int q = 0; q < bernouliInterval; q++)
            {
                //see if a bike has arrived
                if (randomVariableGenerator[0](generator)) X[i]++;

                //distribute the bikes to any clients waiting
                while (!line.empty() && X[i] > 0)
                {
                    auto client = line.front();
                    line.pop(); //remove from queue

                    X[i]--; //decrement bike count
                }

                //see if a client of class 1-3 has arrived
                for (int j = 1; j <= 3; j++)
                {
                    if (randomVariableGenerator[j](generator))
                    {
                        if (j == 3) totalMoney += 1.25; // class 3 pays per ride

                        //if a client arrives and there are no bikes
                        if (X[j] == 0)
```

```cpp
                        {
                            //add the client into the queue
                            line.emplace(Client{ j });
                            //we apply a penalty, for class3 penalty is 0
                            totalMoney += clientPenalty[j];
                        }
                        else
                        {
                            X[j]--; //otherise just give the client a bike
                        }
                    }
                }
            }
        }

        std::cout << "Total Money at the end of experiment " << totalMoney << std::endl;
        averageMoneyAmount += totalMoney;
    }

    std::cout << "Average amount of money over " << numberOfTrials << " iterations" << " : "
        << (averageMoneyAmount / numberOfTrials) << std::endl;

    return 0;
}
```