```
#ifndef NEWTONLAWOFCOOLING_H_INCLUDED
#define NEWTONLAWOFCOOLING_H_INCLUDED
#include <cmath>
#include <iostream>
class newtonLawOfCooling{
    public:
        //newtonLawOfCooling();// MAY NOT NEED AS DEFAULT CONSTRUCTOR
        newtonLawOfCooling(double, double, double, double, double);
        void calcTime1();
        double H(double);
        void calcK();
        double timeAtTemp(double);
        double getTimeOfDeath();
    private:
        double a;
        double h_0;
        double h_t1;
        double h_t2;
        double t1;//need to solve using calcTime1(), time that passed after corpse died
        double K;// need to solve using t1 and calcK()
        double dt;// time interval is in minutes
        double maxTime;
};
// constructors
newtonLawOfCooling::newtonLawOfCooling(double wareHouseTemp = 33.33, double tempAtDeath =
98.88, double tempWhenFound = 91.11, double tempAfterTime = 66.66, double dt = 120){//
warehouse temperature, initial temperature, temperature at t=t1, temperature at t=t2,
time between t1 and t2
    this -> maxTime = 0;//this is calculated later
    a = wareHouseTemp;// 33.33
    h_0 = tempAtDeath;// 98.88
    h_t1 = tempWhenFound;// 91.11
    h_t2 = tempAfterTime;// 66.66
    this->dt = dt;// 120 minutes
    calcTime1();
    calcK();
}

// methods
//ANALYTICAL
SOLUTION-------------------------------------------------------------------------
void newtonLawOfCooling::calcTime1(){// calculate t1
    if(h_0 == a){
        //there's a problem
        std::cerr << "building temperature and initial temperature are the same" << std::
endl;
        t1=0;
        return;
    }
     t1 = log((h_0-a)/(h_t2-a));
     t1 /=  log((h_0-a)/(h_t1-a));
     t1 -= 1;
     std:: cout << t1 << std::endl;
     t1 = dt/t1;//done
}

void newtonLawOfCooling::calcK(){// calculate K, using t1
    if(t1==0){
        std::cerr<< "time found missing"<<std::endl;
        K=0;
        return;
    }
    K = std::log((h_0-a)/(h_t1-a))/t1;
}

double newtonLawOfCooling::timeAtTemp(double h){
```

```
    if(K==0 || h==h_0)
        return 0;
    return std::abs(log((h_0-a)/(h-a))/K);
}

double newtonLawOfCooling::H(double t){
    return exp(-K*t)*(h_0-a)+a;
}

double newtonLawOfCooling::getTimeOfDeath(){
    return t1;
}

//NUMERICAL
SOLUTION-----------------------------------------------------------------------------------


#endif // NEWTONLAWOFCOOLING_H_INCLUDED
```