Application 2 - Object Oriented C++

José Mauricio Sevilla[1]
Andrés Camilo Sevilla[1]

[1]Universidad Nacional de Colombia

5 of March of 2016

Introduction

In this presentation we want to show a different way to program, thinking different, more applied to the physical reality than before.

This thinking is called *Object Oriented Programming*.

We are gong to present some of the basic concepts of this kind of programming, and then applying them to a one specific example.

In physics, any system we study has associated some characteristics, for example, mass, size, density, etc, that define all the system, sometimes, we are interested only in some of them, and ignore the rest.

The principle idea of the *Object Oriented Programming* is follow this philosophy.

so, first, we have to understand what an *object* and a *class* are.

Definition 1

A Class is the definition of the system, for example, a spheric particle, and their characteristics are, radio, density, charge[a]

[a] but, those characteristics doesn't have any value, just say that the spherical particles must have those properties.

Definition 2

An Object is the most basic concept of the Object Oriented Programming, is the class and the specification of a particular object, i.e., having the characteristics specified.

Those definitions could be very strange, but, we'll understand them better doing an example.

For example, we can think about pendulums as the class, they have some characteristics as mass and length, and we can think about the object as a particular pendulum, one with length $1{,}0m$ and mass $50{,}0g$.

To get this concept a little bit clearer, let's make an example of how to make the definition.

Example

Let's construct a class called `Circle`, and it'll have two characteristics, color and radius. There are two kind of variables in the classes, but we aren't interested in them yet,

```
class Circle {
private:
    double radius;
    string color;
public:
    double getRadius();
    double getArea();
}
```

In here, `getRadius()` and `getArea()` are functions defined inside the class, we also must specify them.

This is a basic definition, but, we need to know how to interact with it, create objects. we will construct the code explaining part by part, so first we must open the file where we want to save it, for example in

```
~Documents/OOP
```

And we open/create the file typing

```
gedit example1.cpp &
```

And now, let's do the code.

As always the first thing we do is declare the libraries we are going to use:

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

Then we declare the class, but now we specify what the functions do.

```
class Circle {
private:
    double radius;
    string color;
public:
    // Initialization
    Circle(double r = 1.0, string c = "red") {
        radius = r;
        color = c;    }
    double getRadius() {
        return radius;    }
    string getColor() {
        return color;    }
    double getArea() {
        return radius*radius*3.1416;    }
};   //<------ It Finishes in ;
```

The last thing left to do, is the `main()` function, and that's the only part we haven't done till now.

```
int main() {
Circle c1(1.2, "blue");
std::cout<<"R="<<c1.getRadius()<<std::endl;
std::cout<<"A="<<c1.getArea()<<std::endl;
std::cout<<"C="<<c1.getColor()<< endl;
return 0;
}
```

And like this, we made the first example of how using the object oriented programming, but, those can be very strange concepts, so, we'll do as much examples as we can.

Pendulum

To apply the concepts we just saw, we are going to use a physical situation, we are going to think about the solution of a pendulum. In the last session, we used a method to solve differential equation, in particular, to solve a harmonic oscillator, in this case, we are going to use another method, the method of *Euler-Cromer*, we won't deduce the method just use it.

Now, we are going to understand the method and the properties our class will have.

We can start form the differential equation to the angle θ for a pendulum

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta) - q\frac{d\theta}{dt} + F_D \sin(\Omega_D t) \tag{1}$$

where g is the gravity, l is the length of the pendulum, q is a friction coefficient and $F_D = \sin(\Omega_D t)$ is an external force. We usually take an small angles approximation, so $\sin(\theta) \sim \theta$, we are not going to take that assumption, and will work with (1).

Just like in the past session, we must write this equation as a system of first order differential equations

$$\frac{d\omega}{dt} = -\frac{g}{l}\sin(\theta) - q\frac{d\theta}{dt} + F_D \sin(\Omega_D t) \tag{2}$$

$$\frac{d\theta}{dt} = \omega \tag{3}$$

Euler-Cromer Method

The *Euler-Cromer* method consist in:

$$\omega_{n+1} = \omega_n - \left[\frac{g}{l}\sin(\theta_n) - q\omega_n + F_D \sin(\Omega_D t_n)\right]\Delta t$$

$$\theta_{n+1} = \theta_n + \omega_{n+1}\Delta t$$

$$t_{n+1} = t_n + \Delta t$$

Implementation

In this case we have some quantities that are constants as the gravity g, π, and the time-space between interactions Δt, in this case we are going to do it using the command `const`

```
//Libraries declaring
#include <iostream>
#include <cmath>
#include <stdlib.h>
//Global variables declaring
const double PI=3.1415;
const double G=9.8;
const double DT=0.04;
```

Here we choose $\Delta t = 0{,}04s$.

Now, the pendulum class is defined

```
//Pendulum Class
class pen
{
public:
   double l;//Length
   double q;//Friction coeficient
   double F;//External force
   double fr;//External force frequency
   double W,T;//angular velocity and Angle
   pen();//initial values.
   ~pen();//Destructor.
};
```

But we have to specify those functions.

Then,the functions in the pendulum class.

```
pen::pen(){
   q=0.0;
   F=0.0;
   fr=0.0;
   l=0.0;
   W=0.0;
   T=0.0;
}
pen::~pen(){
}
```

Then we declare two functions, the first change the initial conditions of the class, and the second, have the Euler-Cromer method

```
//Function declaring
void initial_conditions(pen & p);
void time_step(pen & p,double dt,double t);
```

Then, start the main() function

```
//Main function
int main(int argc, char** argv){
  int n;//Iteration number
  pen p,q;//Two pendulum are defined
  double t;
  double b=0.0;
  double a=0.0;
  double dte=0.001;
  n=atoi(argv[1]);//is recieved from the terminal
  // intialization
  initial_conditions(p);
  initial_conditions(q);
  q.T+=dte;   t=0.0;
  for (int i=1; i<=n; i++){
    time_step(p,DT,t);
    time_step(q,DT,t);
    a=fabs(q.T-p.T);
std::cout<<t<<'\t'<<p.T<<'\t'<<p.W<<'\t';
std::cout<<q.T<<'\t'<<q.W<<'\t'<<a<<std::endl;
    t+=DT;  }
  return 0;}
```

The last thing we have to declare, is the functions Initial_Conditions, and time_step

```
void time_step(pen & p,double dt,double t)
{  // Euler- Cromer
p.W+=(-(G/p.l)*sin(p.T)+p.q*p.W-p.F*sin(p.fr*t))*dt;
  p.T+=p.W*dt;
  if(p.T < -PI){
    p.T+=2*PI;
  }else if(p.T > PI){
    p.T+=-2*PI;
  }}
void initial_conditions(pen & p)
{
  p.q=-0.2;
  p.F=0.5799;
  p.fr=0.6;
  p.l=9.8;
  p.W=0.2;
  p.T=0.0;
}
```

Now, let's save it, compile and run.

```
g++ pendulum.cpp -o example2
./example2 > data2.dat
```

Let's make the plots and and interpret the results.