Abstraction in Classes

This lesson will define what abstraction is and how it relates to data hiding.

WE'LL COVER THE FOLLOWING

Abstraction is the second component of data hiding in OOP. It is an extension of encapsulation and further simplifies the structure of programs.

What is Abstraction?

> Abstraction focuses on revealing only the relevant parts of the application while keeping the inner implementation hidden.

Users will perform actions and expect the application to respond accordingly. They will not be concerned with how the response in generated. Only the final outcome is relevant.

There are countless real life examples which follow the rules of abstraction. Take the *Volume* button on a television remote. With a click of a button, we request the T.V. to increase its volume. Let's say the button calls the `volumeUp()` function. The T.V. responds by producing a sound larger than before. How the inner circuitry of the T.V. implements this is

oblivious to us, yet we know the

exposed function needed to interact
with the T.V.'s volume.

Another instance of abstraction is our daily use of vehicles. To our general knowledge, the race peddle tells the car to consume fuel and increase its speed. We do not need to understand the mechanical process happening inside.

Class Abstraction

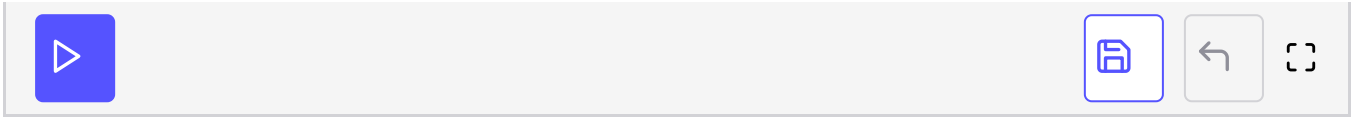
So, let's put all this theory into practice. In the code below, we have a basic class for a circle:

```
class Circle{
  double radius;
  double pi;
};
```

It has two variables, `radius` and `pi`. Now let's add the constructor and functions for the area and perimeter:

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

class Circle{
  double radius;
  double pi;

  public:
  Circle (){
    radius = 0;
    pi = 3.142;
  }
  Circle(double r){
    radius = r;
    pi = 3.142;
  }

  double area(){
    return pi * radius * radius;
  }

  double perimeter(){
    return 2 * pi * radius;
  }
};
```

```
int main() {
  Circle c(5);
  cout << "Area: " << c.area() << endl;
  cout << "Perimeter: " << c.perimeter() << endl;
}
```



As you can see, we only need to define the radius of the circle in the constructor. After that, the `area()` and `perimeter()` functions are available to us. This interface is part of encapsulation. However, the interesting part is what happens next.

All we have to do is call the functions and voila, we get the area and perimeter as we desire. Users would not know what computations were performed inside the function. Even the `pi` constant will be hidden to them. Only the input and the output matter. This is the process of **abstraction using classes**.

In the next lesson, we will explain the second technique of abstraction: **abstraction using header files**.