What Is Abstraction?

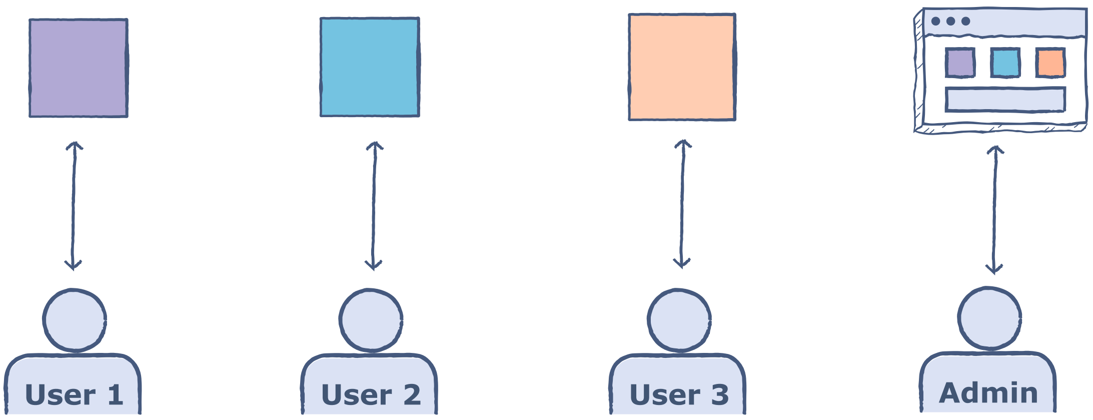
This lesson will go over with a very important concept of OOP named abstraction.

WE'LL COVER THE FOLLOWING

Definition

Abstraction refers to exposing only the essential features of an object to the user and hiding the inner details to reduce complexity. It can be put this way: the user only has to know *what an object does?* rather than *how it does it?*.

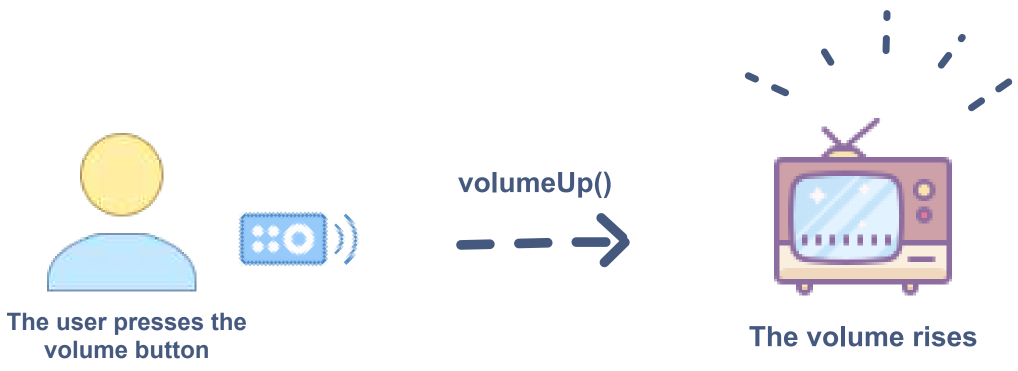
Real-world Examples



The above illustration of the *users* and the *admin* of an application is a good real-world example of **abstraction**.

- A *user* can only use and interact with the limited features of an application, i.e., the user interface, and is unaware of the implementation details or the way the application was developed. Usually, the users are only concerned with the functionality of an application.
- An *admin* can have the access to a lot more features of the application and nothing is hidden from him. The admin can monitor the activity of the users, knows how the application was developed and can implement new features by deploying them in the application.

In the above example, abstraction is being applied to the user but not to the admin.



Let's look at another example 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 unknown to us, but we do know the exposed function needed to interact with the T.V.'s volume.

An Example from C#

In C#, one can easily experience abstraction in action. Let's look at example of C# System.Math class. There are a lot of built-in methods in this class which are known to the developers and can be used by them to get facilitated. Let's use a few methods in our code to access the built-in functionality:

```
class TestAbstraction {

  public static void Main(string[] args) {
    int min = Math.Min(15,18);     //find min of two numbers
    double square = Math.Pow(23,2); //calculate the power of a number
    Console.WriteLine("The min of 15 & 18 is: " + min);
    Console.WriteLine("The square of 23 is: " + square);
  }

}
```

In the above code:

- The `Math.Min()`, a static method, is used to find the minimum of the two numbers.
- The `Math.Pow()`, a static method, is used to find 23 to the power 2.

But the user doesn't have to know about the implementation of these two methods inside the Math class.

Abstract Data Types

> An abstract data type (or class) is a type that defines *what operations are to be performed?* rather than *how they are to be performed?*

By the definition of abstract data types, the users only get to know the essentials or the functionality of those data types. What goes on behind the scenes in getting the implementation to achieve the specified functionality is hidden.

An example of `abstract` data types can be the built-in Stack class in C# for which the user knows that it has the `Push()`, `Pop()`, `Peek()` etc. methods but doesn't know how these are implemented.

How to Achieve Abstraction

In C#, we can use the following components to achieve abstraction:

- *Abstract Classes*
- *Interfaces*

These two topics will be covered in the upcoming lessons of this chapter.

The `abstract` Modifier

The `abstract` Modifier

In C#, it is nearly impossible to achieve abstraction without using the `abstract` modifier. This modifier indicates that the thing being modified has a missing or incomplete implementation.

Whenever the modifier `abstract` is used with classes or methods, those methods or classes only specify *what operations should be done*. Whoever uses these methods or classes in their code will have to deal with the implementation details of these methods or classes. Isn't the definition of abstraction the same, to tell someone what should be done? How it is to be done is another story.

Now that we are done with the basics of abstraction, the next lesson covers the topic of abstract classes and methods.