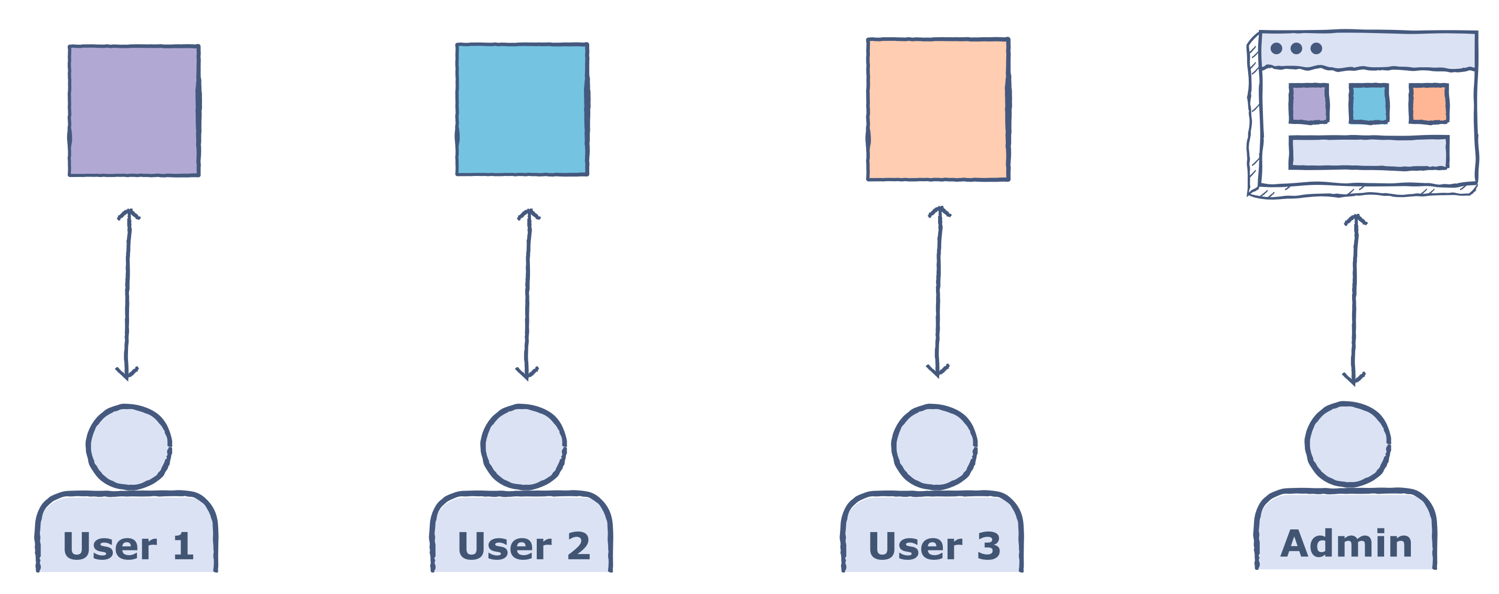
Definition

Abstraction in Object-Oriented Programming refers to showing only the essential features of an object to the user and hiding the inner details to reduce complexity. It can be put this way that the user only has to know *“what an object does?”* rather than *“how it does?”*.

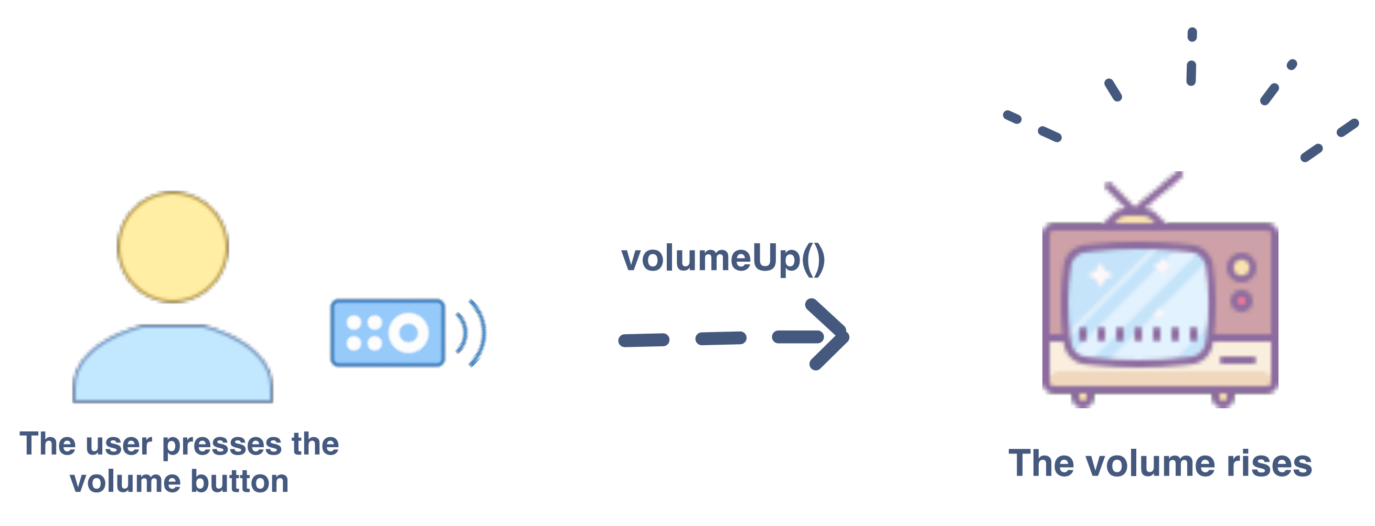
Real-world Examples



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



Let’s look into 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 oblivious to us, yet we know the exposed function needed to interact with the T.V.'s volume.

An Example from Java

In Java, one can very easily see abstraction in action. Let’s take an example of Java Math class. There are a lot of in-built methods in this class that can be used by the programmer to get facilitated. Let’s use a few methods in our code to access the in-built functionality:

```java
class TestAbstraction {

  public static void main( String args[] ) {
    int min = Math.min(15,18);      //find min of two numbers
    double square = Math.pow(2,2); //calculate the power of a number

    System.out.println("The min of 15 & 18 is: " + min);
    System.out.println("The square of 2 is: " + square);
  }

}
```

Run Save Reset

In the above code:

- The `Math.min()` method is used to find the minimum of the two numbers
- The `Math.pow()` method is used to find 2 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 only defines *‘what operations are to be performed?’* rather than *‘how to be performed?’*

By the definition of abstract data types, the users only get to know the essentials i.e. the functionality of those data types and ‘how the implementation should be done to achieve the specified functionality?’ part is hidden.

An example of `abstract` data type can be a built-in stack class in Java for which the user knows that it has the `push()`, `pop()`, `size()` e.t.c. methods but doesn’t know how these are implemented.

How to Achieve Abstraction

In Java, we 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` Keyword

In Java, it is impossible to achieve abstraction without using the `abstract` keyword. The abstract keyword can be used with the *methods* and *classes* only.

Whenever the keyword `abstract` is used with the classes or methods, those methods or classes then only specify *“what operations should be done”* and whoever will use this method or class in their code will have to deal with the implementation details of this method or class. Isn’t the definition of abstraction the same? i.e. to tell someone what should be done and how it is to be done is another story.