Abstraction is the process of making complex ideas into simple ones, removing the characteristics less important and focusing on the essential ones. It's a programming concept where we define a set of methods and properties behaviors based on a class without specifying how they are or will be implemented, so we can create a blueprint for a class focusing on the more important characteristics of the program. As mentioned before the use of abstraction helps us to focus on the essential characteristics of the program, for example, creating abstract classes helps us to have a template of our program that will guide us throughout its implementation, this template will reduce the time spent on creating a code without any planning which would cause us to keep thinking about what is the next step or how the program must be formulated, instead of implementing the code based on a blueprint already formulated.

 For example, let's say we want to create a complex program that will be responsible to store the car's rims for a mechanical workshop, the users will be able to ask for their rims and their size. We will break this program into two classes, the class Rims will be responsible to store all the rims data(name and size) and compare it to the user entries and the Entries class will be responsible to get the entries (the users' input) and store them in the Rims class. In this example, we didn't specify using actual code how we will do it, but we established behaviors(things that will be performed) for those classes, which can give us a blueprint about how the program will be formulated.

Now we can start coding the program based on its behaviors and classes:

**using System;**

**using System.Collections.Generic;**

// 1- Storing data and comparing it to user's entries(Create a filename.cs file to do it)

**class Rims {**

**public List<string> rimsList = new List<string>();**

**public void AddRim(string name, string size) {**

**rimsList.Add($"{name}: {size}");**

**}**

**}**

// 2- Comparing user input with the rims inside list(Create a filename.cs file to do it)

**class Entries {**

**bool found = false;**

**public void CompareRims(List<string> rims, string userInput) {**

**foreach(string rim in rims){**

**if (rim == userInput) {**

**found = true;**

**break;**

**}**

**}**

**if (found) {**

**Console.WriteLine("We have this rim in our stock.");**

**}**

**else {**

**Console.WriteLine("We don't have this rim in our stock.");**

**}**

**}**

**}**

// 3- Starting the program

**class Program {**

**static void Main(string[] args) {**

**// Adding random rims so they can be compared to the user input**

**Rims myRims = new Rims();**

**myRims.AddRim("Rim A", "16");**

**myRims.AddRim("Rim B", "17");**

**myRims.AddRim("Rim C", "18");**

**// Create a new Entries object and compare user input with the rims list**

**Entries myEntries = new Entries();**

**List<string> rimsList = myRims.rimsList;**

**Console.Write("Enter the name of the rim: ");**

**string userRim = Console.ReadLine();**

**Console.Write("Enter the size of the rim: ");**

**string userSize = Console.ReadLine();**

**string userInput = $"{userRim}: {userSize}";**

**myEntries.CompareRims(rimsList, userInput);**

**}**

**}**

The example above demonstrates how we can apply the principle of abstraction. It's easier to organize the elements of the code using the abstract principle, we can do the code faster following a blueprint created by us based on the behavior of the classes.