Abstraction

Abstraction is a common practice in programing where complex tasks which takes many lines of code to achieved is compressed down into a single or few lines of codes which then can be used and re-used throughout an entire project. In general, it is a way of making complex tasks simple.

Some benefit of abstraction include:

1. It saves the programmer or the developer’s time. Tasks that require many lines of code is compressed into a single line of code and re-used anytime those lines of codes are needed.
2. It also helps programmers make their code readable. For example, line of code that computes or calculate the area of a shape is converted into a function named “calculateArea.” Anytime another programmer reviews the code, he or she knows that that line of code is meant for calculating the area.

The following code shows how abstraction can be very helpful.

Code Example:

abstract class Animal {

public abstract void MakeSound();

public void Sleep() {

Console.WriteLine("Zzz");

}

}

class Dog : Animal {

public override void MakeSound() {

Console.WriteLine("Woof");

}

}

class Program {

static void Main(string[] args) {

Dog myDog = new Dog();

myDog.MakeSound();

myDog.Sleep();

}

}

Explanation:

1. In the example above, the Animal class is an abstract class that contains an abstract method MakeSound() and a regular method Sleep().
2. The MakeSound() method is abstract, meaning it does not have a body and must be implemented by any subclass that inherits from Animal.
3. The Dog class extends Animal and provides the implementation for the MakeSound() method.
4. This way, the Animal class abstracts the concept of an animal making a sound, while the Dog class provides the specific implementation for a dog.
5. In our program, we can refer to these complex methods by simply calling them as follows,
   * Dog myDog = new Dog(); “To create a new Dog object.”,
   * myDog.MakeSound(); “call the abstract method.”,
   * myDog.Sleep() “call the regular method.”;