**Design a Workflow engine :**

Design a workflow engine that takes a workflow object and runs it. A workflow is a series of steps

or activities. The workflow engine class should have one method called **Run()** that takes a

workflow, and then iterates over each activity in the workflow and runs it.

We want our workflows to be extensible, so we can create new activities without impacting the

existing activities.

Educational tip: we should represent the concept of an activity using an interface. Each activity

should have a method called **Execute()**. The workflow engine does not care about the concrete

implementation of activities. All it cares about is that these activities have a common interface:

they provide a method called **Execute().** The engine simply calls this method and this way it

executes a series of activities in sequence.

The aim of this exercise is to help you understand how you can use interfaces to design

extensible applications. You change the behaviour of your application by creating new classes,

rather than changing the existing classes. You’ll also see polymorphic behaviour of interfaces.

**Real-world use case: in a real-world application you may use a workflow in a scenario like the**

**following:**

1- Upload a video to a cloud storage.

2- Call a web service provided by a third-party video encoding service to tell them you have a

video ready for encoding.

3- Send an email to the owner of the video notifying them that the video started processing.

4- Change the status of the video record in the database to “Processing”.

Each of these steps can be represented by an activity. For the purpose of this exercise, do not

worry about these complexities. Simply use Console.WriteLine() in each of your activity classes.

Your focus should be on sending a workflow to the workflow engine and having it run the

workflow and all the activities inside it. We don’t care about the actual activities.