Advanced Programming Techniques (java or c #)

1. Interfaces and replacing conditional with polymorphism

Copy this code into C# console application. Make sure you can run it.

```
internal class Program
{
   static bool runs = true;


   private static void Main(string[] args)
            bool runs = true;
            while (runs)
            {
               string? command = Console.ReadLine();
               if (command == "exit")
               {
                  runs = false;
               }
               else if (command == "sum")
               {
                  int firstTerm = int.Parse(Console.ReadLine());
                  int secondTerm = int.Parse(Console.ReadLine());

                  Console.WriteLine(firstTerm + secondTerm);
               }
               else if (command == "dif")
               {
                  int firstTerm = int.Parse(Console.ReadLine());
                  int secondTerm = int.Parse(Console.ReadLine());

                  Console.WriteLine(firstTerm - secondTerm);
               }
               else
               {
                  Console.WriteLine("Unknown command");
               }
            }
      }
}
```

Replace each branch of the if-else statement with implementation of the Operation interface. Interface returns *string,* so print the operation result in the main file. Skip the "exit" operation for now.

```
   interface Operation
   {
      string perform();
   }
```

You can store instances of the Operation in the dictionary and use command names as keys.

```
var commands = new Dictionary<string, Operation>();
commands[Console.ReadLine()];
```

2. Delegates

Add an Exit implementation of the Operation interface. Add an Action delegate to it and call it in perform() function. In the main program add static ExitProgram that flips *runs* flag and pass it to the Operation instance. You can return “Program end” as the result. Commit your solution.

3. Anonymous functions and more delegates

Replace Exit Program with lambda expression. Change signature of the *string perform()* function to void perform(). Create a following delegate

```
public delegate void OnResult(string result);
```

Add it in each of the Operation implementations and use it instead of returning a value. Pass a *Console.WriteLine* during instance creation.

Create and use another delegate to remove *Console.ReadLine* from Operations.