**Exercise 1**

Step1 :Create an LSP violation Lab project

Step 2: create a bird interface

|  |
| --- |
| internal interface Bird  {  void Fly();    } |

Step 3: create multiple concrete birds which can fly

|  |
| --- |
| internal class Parrot : Bird  {  public void Fly()  {  Console.WriteLine("Parrot is Flying");  }  }  internal class Pigeon : Bird  {  public void Fly()  {  Console.WriteLine("Pigeon is Flying");  }  } |

Step 4:Here we add class Ostritch which throws exception in fly method by making a false promise

|  |
| --- |
| internal class Ostrich : Bird  {  public void Fly()  {  throw new Exception("But Ostrich cannot Fly");  }  } |

Step 5: Create a program class and test the application and see how substitution principle is violated.

|  |
| --- |
| internal class Program  {  static void Main(string[] args)  {  Parrot parrot1 = new Parrot();  Pigeon pigeon1 = new Pigeon();  Ostrich ostrich1 = new Ostrich();  StartFlying(new Bird[] {parrot1,pigeon1,ostrich1 });  }  private static void StartFlying(Bird[] birds) {  Console.WriteLine( "birds starts flying");  foreach (Bird bird in birds)  {  bird.Fly();  }  }  } |

**Exercise 2**

Step1 :Create an LSP Solution Lab project

Step2:Create an interface Flyable

|  |
| --- |
| internal interface Flyable  {  void Fly();  } |

Step3:Make concrete types as show below

|  |
| --- |
| internal class Parrot :Flyable  {  public void Fly()  {  Console.WriteLine("Parrot is Flying");  }  }  internal class Pigeon : Flyable  {  public void Fly()  {  Console.WriteLine("Pigeon is Flying");  }  }  internal class Ostrich  {    } |

Step4:Make program class as shown below

|  |
| --- |
| internal class Program  {  static void Main(string[] args)  {  Parrot parrot1 = new Parrot();  Pigeon pigeon1 = new Pigeon();  Ostrich ostrich1 = new Ostrich();  StartFlying(new Flyable[] { parrot1, pigeon1 });  }  private static void StartFlying(Flyable[] birds)  {  Console.WriteLine("birds starts flying");  foreach (Flyable bird in birds)  {  bird.Fly();  }  }  } |