Exercise 1:

Step 1: Create an ISPViolationApp with following interfaces and derivatives

|  |
| --- |
| internal interface IWorker  {  void StartWork();  void StopWork();  void StartEat();  void StopEat();  }  internal class Manager : IWorker  {  public void StartEat()  {  Console.WriteLine("Manager Starts eating");  }  public void StartWork()  {  Console.WriteLine("Manager Starts Working");  }  public void StopEat()  {  Console.WriteLine("Manager Stops eating");  }  public void StopWork()  {  Console.WriteLine("Manager Stops Working");  }  }  internal class Robot : IWorker  {  public void StartEat()  {  throw new Exception("Robots Cannot Start Eating");  }  public void StartWork()  {  Console.WriteLine("Robot Starts Working");  }  public void StopEat()  {  throw new Exception("Robots Cannot Stop Eating");  }  public void StopWork()  {  Console.WriteLine("Robot Stops Working");  }  } |

Step2: test the application from Program.cs and see where is substitution principle getting violated and figure out how to solve this problem by splitting interfaces or ISP

|  |
| --- |
| {  static void Main(string[] args)  {  var bot = new Robot();  var manager = new Manager();  AtTheWorkStation(manager);  AtTheWorkStation(bot);  AtTheCafeteria(manager);  AtTheCafeteria(bot);  }  private static void AtTheCafeteria(IWorker worker)  {  Console.WriteLine("At the Cafeteria");  worker.StartEat();  worker.StopEat();  Console.WriteLine();  }  private static void AtTheWorkStation(IWorker worker)  {  Console.WriteLine("At the Work Station");  worker.StartWork();  worker.StopWork();  Console.WriteLine();  }  } |

Exercise 2:

Step 1:Make a ISPRefactoredApp and split interfaces as given below

|  |
| --- |
| internal interface IEat  {  void StartEat();  void StopEat();  }  internal interface IWork  {  void StartWork();  void StopWork();  }  internal interface IWorkEat : IWork,IEat  {  } |

Step2:Make derivatives like below

|  |
| --- |
| {  public void StartEat()  {  Console.WriteLine("Manager Starts eating");  }  public void StartWork()  {  Console.WriteLine("Manager Starts Working");  }  public void StopEat()  {  Console.WriteLine("Manager Stops eating");  }  public void StopWork()  {  Console.WriteLine("Manager Stops Working");  }  }  internal class Robot : IWork  {  public void StartWork()  {  Console.WriteLine("Robot Starts Working");  }  public void StopWork()  {  Console.WriteLine("Robot Stops Working");  }  } |

Step3: Now test the application from program.cs ,note that now at Cafeteria now robots are not allowed as we made correct design using ISP

|  |
| --- |
| static void Main(string[] args)  {  var bot = new Robot();  var manager = new Manager();  AtTheCafeteria(manager);  //AtTheCafeteria(bot);  AtTheWorkStation(manager);  AtTheWorkStation(bot);  }  private static void AtTheCafeteria(IEat eater)  {  Console.WriteLine("At the Cafeteria");  eater.StartEat();  eater.StopEat();  Console.WriteLine();  }  private static void AtTheWorkStation(IWork worker)  {  Console.WriteLine("At the Work Station");  worker.StartWork();  worker.StopWork();  Console.WriteLine();  } |

//