**Single Responsibility:**

* Open Demos>Java>SRP Example in IntelliJ and navigate to src>com>solid
* Explain that the User class is a POJO (Plain Old Java Object)
  + All it does is store attributes and creates users
  + Does not extend prespecified classes or interfaces, and does not contain prespecified annotations
* Show the UserSettingService class and explain that it contains the methods for changing a user’s information
  + Currently we have a method for changing the user’s email
  + Since we need to check that the user is a valid user before attempting to change the user’s email, we also have a checkAccess method to validate the user
  + This class has 2 responsibilities, it changes user settings, and it validates the user
  + If we need to validate the user in another class later, we’ll have to use the UserSettingsService class to do so or recreate the checkAccess method in the new class
* Create a new class called SecurityService
* Move the checkAccess method into it, and call checkAccess via the SecurityService in UserSettingsService
* Now we can validate the user from any class using the SecurityService, and the UserSettingService has only one responsibility – handling user settings

**Open-Closed Principle:**

* Open Demos>Java>OCP Example in IntelliJ and navigate to src>com>solid
* Show students that the Rectangle class is an extremely simple POJO that only holds the rectangle’s length and width attributes
* Show that the AreaCalculator contains a method for calculating the area of a rectangle
* Create a Circle class that hold a radius attribute
* Add a method in AreaCalculator to calculate the area of a circle (calculateCircleArea)
* Tell the students that currently, every time we create a new shape class, we have to modify AreaCalculator to be able to calculate the area of that class
  + Any consumers of this class will have to update their libraries every time we modify AreaCalculator
* Instead, create a Shape interface that includes the empty method calculateArea
* Make Rectangle and Circle implement the Shape interface – this will give you an error because they are not overriding the calculateArea method, so add that now
  + Rectangle – length \* width
  + Circle – Math.PI \* (radius \* radius)
* Now area calculator only needs one method – calculateShapeArea which takes a Shape and returns shape.calculateArea()
* Create a new Triangle class to show that nothing needs to be modified when we add a new class, we can simply extend Shape
  + (base \* height) / 2

**Liskov Substitution:**

* Open Demos>Java>LSP Example in IntelliJ and navigate to src>com>solid
* Walk students through the Bird interface that says birds should all be able to fly, walk, and lay eggs
* Show that the Duck class implements the Bird interface and defines how a duck flies, walks, and lays eggs
  + A duck is a bird
* Show that the Ostrich class also implements the Bird interface but only defines how the ostrich walks and lays eggs, because ostriches can’t fly
  + This gives us an error because classes must override all of the methods in the class’s interface (error not visible in GitHub VS code IDE)
* Create a FlightlessBird interface that has the methods walk and layEggs
* Update Ostrich to implement the FlightlessBird interface – note that the error goes away
  + Though an ostrich is a bird, it cannot replace a bird, especially our implementation of a bird that requires flight

**Interface Segregation:**

* Open Demos>Java>ISP Example in IntelliJ and navigate to src>com>solid
* Show students the MultiFunctionPrinter interface and all of the methods it contains
* Switch over to show the XeroxWorkCenter which implements the MultiFunctionPrinter interface and overrides all of its methods
* Show that the HPPrinterAndScanner class also implements the MultiFunctionPrinter interface and overrides all the methods but the fax method is empty because this printer cannot send faxes
* Last, show that that the CannonPrinter class also implements the MultiFunctionPrinter interface and overrides all the methods but the only method that contains a code block is the print method because this printer cannot scan or send faxes
  + Someone using this system could try to use the CannonPrinter, seeing that it has a scan method, to scan and the code will break
* Rename the MultiFunctionPrinter interface to Printer
* Create an interface named Scanner and move the scanning methods from Printer to this interface
* Create an interface named FaxMachine and move the fax method from Printer to this interface
* Make XeroxWorkCenter implement all 3 interfaces
  + Note that XeroxWorkCenter is a printer, scanner, and fax machine
* Make HPPrinterAndScanner implement Printer and Scanner
  + Note that HPPrinterAndScanner is a printer and scanner
* Make CannonPrinter implement Printer
  + Note that CannonPrinter is a printer only
  + Now CannonPrinter and HPPrinterAndScanner are no longer forced to implement methods they do not use

**Dependency Inversion:**

* Open Demos>Java>DI Example in IntelliJ and navigate to src>com>solid
* Show the students the Project class where we are mocking a development project that has a fontend developer that writes javascript, and a backend developer that writes java
* Quickly show that the FrontendDeveloper and BackendDeveloper classes each have a different method for writing in their respective languages
  + Currently this project depends on having a single backend dev who writes java, and a single frontend dev who writes javascript
  + If the frontend dev switched to HTML, we have to change the method in FrontendDeveloper and its call in Project
* Create an interface called Developer that contains the method develop()
* Make FrontendDeveloper implement Developer and call writeJavaScript() in develop()
* Make BackendDeveloperimplement Developer and call writeJava() in develop()
* Give Project a List of Developers attribute called developers
* Create a constructor for Project that takes a List of Developers and sets the developers attribute to said List
* Switch implement() to loop through the list of developers and call develop() on each developer in the list
  + Now we can add as many developers of any type to the project as needed
  + The project doesn’t know what kinds of devs are on the project and what language they’re writing in, it just has a list of developers and tells them to develop