**Adapter:**

* ​​Open Demos>Java>adapter in IntelliJ and navigate to src>main>java
* Walk students through the methods declared in Phone, LightningPhone, and USBCPhone
* Show students how methods have been overridden in Android and IPhone
* In Main, create a method called chargePhone that takes a USBCPhone and returns void
  + In chargePhone call phone.plugInUSBC and phone.charge()
* In the main method, create new instances of IPhone and Android
* Call chargePhone 2 twice and pass in the instances of Android and IPhone
* You should get a compiler error that does not allow you to pass an IPhone into chargePhone
* To correct this, create a new class called LightningToUSBCAdapter that implements USBCPhone
  + Give it a field of type IPhone
  + Generate a constructor that takes an IPhone and assigns it to the iPhone field
  + Override charge() to print phone + “ is charging.”
  + Override plugInUSBC to print this + “ plugged into USB Cable.” and this + “ plugged into “ + phone
* In Main, create an instance of LightningToUSBCadapter and pass in the iPhone
* Pass the adapted iPhone into chargePhone – run the program

**Decorator:**

* Open Demos>Java>decorator in IntelliJ and navigate to src>main>java
* Walk students through the Coffee abstract class and its DripCoffee implementation
* Create an abstract CoffeeDecorator class that extends Coffee
  + Give it a private final Coffee field
  + Generate a protected constructor that takes the description, cost, and coffee
  + In the constructor, call super(description, cost) and set this.coffee equal to the passed in value of coffee
  + Override getCost to return coffee.getCost() + this.cost
  + Override getDescription to return String.format("%s\nwith %s", coffee.getDescription(), description)
* Create a new class called Cream that extends CoffeeDecorator
  + Create and initialize a private static final String for the description and a private static final double for the cost
  + Generate a constructor that takes an instance of Coffee and calls super(DESCRIPTION, COST, coffee)
* In Main, create a String to hold the message format, "Your %s\nis ready for pick up for %.2f\n\n“
  + Create a new instance of Cream and pass it a new instance of DripCoffee
  + Format print the message format with the coffee’s description and cost – run the program
* Create the new classes Syrup , and WhippedCream that extend CoffeeDecorator
* In Main, create an instance of WhippedCream and pass in an instance of Cream, which receives an instance of Syrup, which receives an instance of DripCoffee
* Format print the message format with the new coffee’s description and cost – run the program

**Façade:**

* ​​Open Demos>Java>façade in IntelliJ and navigate to src>main>java
* Walk students though the simple methods in Screen, PopcornPopper, Lights, Projector, StreamingPlayer, and SurroundSoundSystem
  + All classes have methods that print a message related to the action being mocked
* Create a new class called HomeTheatre
  + Compose the class with private final instances of all the other classes
  + Generate a constructor that takes in instances of all if the other classes and set the fields to those instances
  + Create a new method called watchMovie that takes a movie title as a String and returns void
  + In watchMovie call popper.on(), popper.pop(), lights.dim(), screen.down(), projector.on(), projector.wideScreenMode(), surroundSound.on(), surroundSound.setVolume(), streamingPlayer.on(), and streamingPlayer.play()
  + Create a new method called endMovie that takes no args and returns void
  + In endMovie, call popper.off(), lights.on(), screen.up(), projector.off(), surroundSound.off(), player.stop(), and player.off()
* In Main, create a new instance of every class
* Call watchMovie then endMovie on the instance of HomeTheatre – run the program